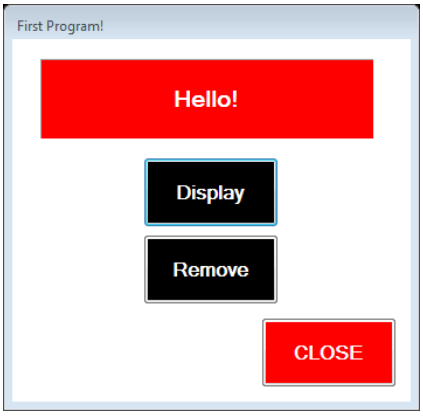
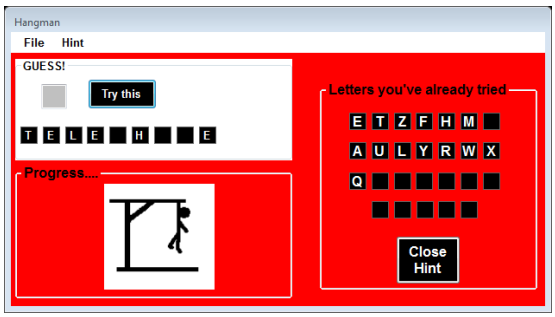


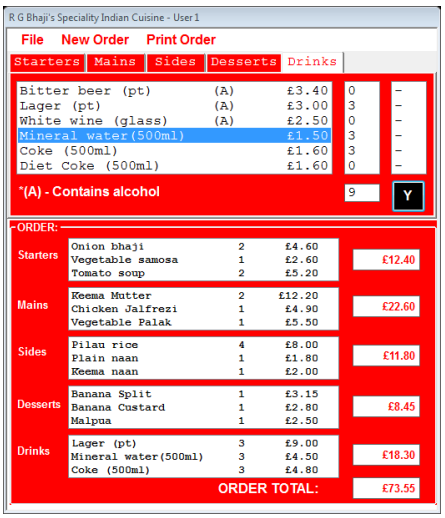
From this.....

Event Driven Programming Using Visual Basic .NET

Lecturer : Jane Fletcher



To this.....



Lecturer: Jane Fletcher

Acknowledgements

To all BTEC National students: past, present and future, each of whom has helped to teach me how to program. Without them I would still be an over-paid Software Consultant with dubious skills, some of which pertained to programming….

I am grateful for your patience while I learned how to teach you to program, and for your guidance when things went wrong and your humour while I stood at the front of the class looking bemused, confused and ever-so-slightly embarrassed….

You have all rocked programming!

Lecturer: Jane Fletcher

Contents

Topic **Page**

Notes:

Introduction to the Unit

Lecturer: Jane Fletcher

This unit is the first one that you will do that includes any programming. It runs over the entire academic year (i.e. 36 timetabled weeks), and each lesson lasts for one hour and a half. You will have to write down lots of program code so it is a good idea to have with you a pen and some lined paper for each of these lessons; later on you may wish to use a computer to take notes, but to start with pen and paper is best. Needless to say, all classes will start exactly on time and if you are late then you will miss some vital explanations and will have to catch up on your own time.

Structure of the classes

During the course of each lesson you will be shown how to write one program; afterwards, you will write the program just demonstrated yourself using your own notes and the program specification available separately on the network and again in this booklet. Once you have done that, and got a bug-free piece of software to demonstrate to your lecturer, you will be asked to move on to write Programming Exercise One for the given week. This program is always based on the one demonstrated in class but is a little more difficult and contains usually one extra skill. Some weeks there may be a second exercise for you to do as well – it depends on how complex the tasks you've been asked to do are. It is important that you complete each of these programs because your skills will build up week-by-week and if you miss out on either a class or writing a program or both, then you may not have the skills you need to complete the next class / program successfully. Use the table in Appendix 3 – "Log of Completed Exercises" – to keep track on your own progress and make sure you get your work signed off by your lecturer.

Every week we will introduce a new object and you will have a document to complete called "Object of the Week". These documents will build up to create a comprehensive manual for you to refer to when you're writing programs independently so it is important that you complete each one, and don't get behind.

Some weeks will include some homework questions for you to complete outside of class time. These questions will help you to build on the practical skills and knowledge that you're gaining by completing the programs set on a weekly basis and again it is important that you complete them since they help you to develop your underpinning knowledge around the concepts of visual programming. If you are set any homework questions, the class will go through them with your lecturer during the next lesson so you get the chance to put right any mistakes you might have made.

There are two assignments for this unit; the first one is theoretical and you will be required to do lots of research in order to complete it. The assignment must include a full bibliography, created using the Harvard Referencing System and without it your work will be rejected. The second assignment is a practical one and requires you to produce a program and all the associated documentation. In order to complete this, you must have done all of the programming exercises starting on page 111 of this book – if you have, you will be able to write the program with little difficulty since all the skills you need will have been already developed.

This unit is one you will either love or hate – keep up with the work, and you'll love it!

Quick start to terminology used in this unit

Lecturer: Jane Fletcher

One of the most difficult things to get to grips with when you are new to programming is the different terminology that is used within a software development environment. The easiest thing to do is to complete a "glossary of terms" like the one given in template form of Appendix 2 of this document. It will help you to get to grips with some of the different uses of common words – when you're a programmer, words have different meanings to the ones you're used to. For example, a "form" when you're not a programmer could be either a wooden bench that you have to sit on when you're in a school gym, or something you have to fill in when you're applying for credit or for a college course. When you're a programmer, a form is an object that is the screen that the user sees when the software is running.

As a "quick-start guide", here is a very brief run-down of some of the most popular terms used within this unit.

Term	Definition
Code	The words that make up the program. The code must follow the correct syntax or the compiler will throw up error messages.
Code view	The section of the program editor that allows the programmer to write the processes for the events belonging to the objects on a form.
Compile	When the program code - written in language understandable to humans - is converted to language understandable by the computer.
Control	A specific visual object that the programmer places on a form, usually at design time.
Crash	The term used within the world of IT meaning an abrupt and unscheduled end to a program's execution due to an unforeseen error.
Data	Collections of information (usually on a related subject) that have been formatted in a specific way. Data is unprocessed information. An example of the difference between data and information is when a user collects answers to a set of questions in a questionnaire and organises them to put into a computer program. That is data. When the user has entered this data into the program and processed it, it becomes information.
Debugger	Part of the programming editor / software that allows the programmer to use various tools to ensure the eventual correct processing of the program. Debuggers allow the programmer to halt execution of the program and to examine the contents of variables / fields.
Design	The part of the program life cycle that comes before coding - putting objects / controls onto a form to meet the designs contained within the program specification.
Design view	The section of the program editor that allows the programmer to put controls / objects onto a form.
Dialog box	A window that appears (usually within the Design view of the Editor) requesting further input from the user. Examples include Font dialog box and the Items Collection Editor of a MenuStrip control.

Quick start to terminology used in this unit

Lecturer: Jane Fletcher

Disabled	A control that is disabled is one that has been deactivated by the programmer and cannot be used by the user at this particular time.
Documentation	The stage of the software design life cycle that involves the programmer writing either technical documentation or user documentation.
Ellipses	The name of the three little dots that appear on some properties seen within the Properties Window. Clicking on the ellipses will generate a Dialog Box.
Enabled	A control that is enabled is one that has not been deactivated by the programmer at this particular time.
Error messages	Either generated by the compiler after compilation or during coding showing errors made by the programmer, or messages coded by the programmer to tell the user that they have done something wrong.
Event	Every object / control has a number of events that can be selected and programmed at design time. Each type of object has a default event: for example, a button object's default is the Click event.
Event handler	The type of event selected for any given control, such as the _Click event of a button.
Form	The window that holds all of the controls that are displayed to the user as a screen. This is the main component of most programming languages, especially visual programming.
Grab handles	The square box shapes that are spaced around the edges of a control to indicate that it is possible to resize this control. To enlarge or shrink the control, click on it to make it the active control (the grab handles will become visible when the control is active) and then click on the grab handle with the left mouse button; move the pointer in the direction you wish to go, then when the control is the correct size release the mouse button.
GUI	Graphical user interface, or the form(s) shown to the user at run time.
Hard-coding	Hard-coding is where the developer codes things like input information or parameters directly into the code, rather than reading it from user input or from a file, or from using things like functions. Examples would be when the developer explicitly writes the path of where a picture can be found into the program rather than using the function Application.StartupPath, or a set of valid passwords for a login rather than reading them from a file. This is generally classed as lazy, sloppy coding and should be avoided where possible.
i-Capping	i-Capping is the convention you must use to name any objects or variables that you use within your projects. i-Capping means using upper case for the first letter of any words used in names, with a prefix in lower case. Examples of this is btnProcessData, intProcesedInformationCount. Notice the prefixes (btn, int) are in lower case but the first letter of each word is in upper case. Any variation on this theme is not acceptable!
Inbuilt function	.NET has a number of functions that come as part of the package. An inbuilt function is a word or phrase that does something specific when used in the correct context within code, and cannot be used in any other context (i.e. function names are reserved words). Examples include the MsgBox() function (which allows the programmer to easily code message boxes and StrConv which changes the format of characters entered by the user to match the requirements of the program (e.g. from a mixture of cases to all upper case).

Quick start to terminology used in this unit

Lecturer: Jane Fletcher

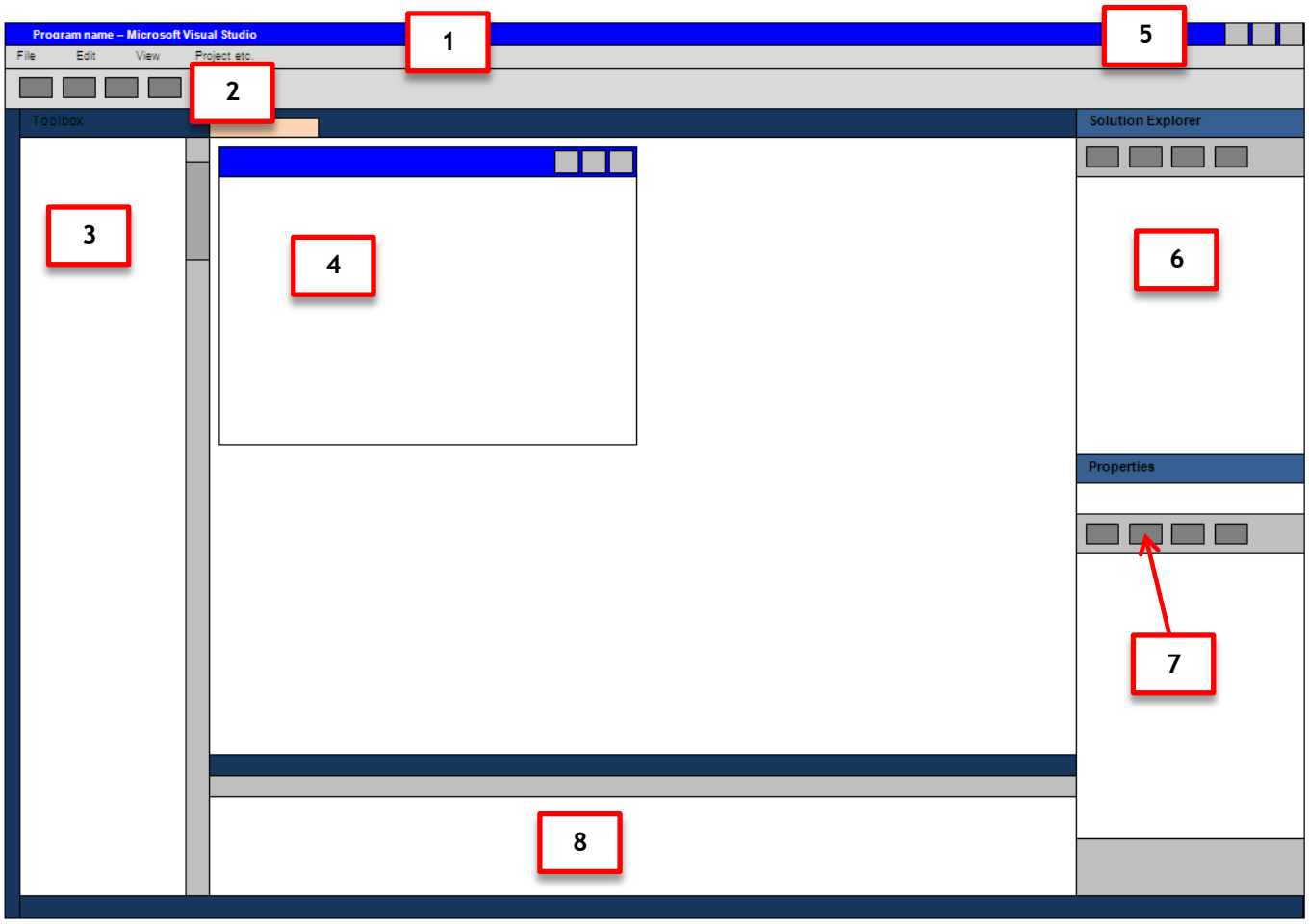
Information	Processed data. An example of the difference between data and information is when a user collects answers to a set of questions in a questionnaire and organises them to put into a computer program. That is data. When the user has entered this data into the program and processed it, it becomes information.
Input	The information that is entered into a program, either electronically from a file or a database, or via a user.
Keyword	See “Reserved word”.
Object	A specific control. The terms "control" and "object" are interchangeable for the purpose of this unit. An object is something that can be individually selected and manipulated.
Output	The information generated by the program after processing input.
Pointer	The "cursor" used at design / code time.
Process	The actions to be performed when a particular event is triggered. Processes for a program should be defined in the program specification by the systems analyst.
Program	A collection of code that performs a specific function. An example program is the Calculator that comes in Accessories.
Program specification	A document written by the systems analyst to tell the programmer exactly what a program (or suite of programs) should look like, and do.
Programming	The act of creating programs, including the initial making of a form, the coding associated with the form and the testing that ensues.
Properties Window	The section of the program editor that allows the programmer to set initial values to the properties of controls used within a form at design time.
Property	A particular attribute that goes towards defining what a control looks like or acts like. Each control has a number of properties. Some properties can be defined / altered at design time, some at run time and some can be defined / altered at both design and run time.
Reserved word	A reserved word is one that has an inbuilt meaning within the .NET framework and cannot be used by the developer for any other purpose. For example, the words “Private” and “Sub” cannot be declared as variable names. If you try you will get an error message – “Keyword is not valid as an identifier”.
Run	The term used as a catch-all and means "compile the program and then make it work."
Run time	The time when the program is executing, either in testing or during full implementation with the user.
Solution Explorer	The section of the program editor that allows the user to navigate between different views of the program (design, code) and between all forms used in the open program.
Storage	The electronic area that has been ring-fenced by the program to be used to store data and to allow the program to execute correctly.
Subroutine	A self-contained section of program code.
Syntax	The acceptable way that program code should be written. Errors in syntax will stop a program compiling and therefore running correctly.
System	For the purpose of this module, a system is a collection of programs that go towards performing a specific function. An example system is a payroll system.

Quick start to terminology used in this unit

Lecturer: Jane Fletcher

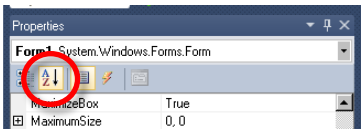
Technical documentation	Written document (online or paper-based) aimed at fellow IT professionals giving necessary details about the current system / program to enable maintenance to take place effectively.
Testing	The stage of the software design life cycle that involves the programmer putting correct and incorrect data into the program and ensuring that a correct outcome ensues.
Toolbox	The section of the program editor that contains all the controls that can be put onto a form.
Trigger function	What decrees which event handler will be run, for example clicking a mouse on a button will cause the associated event handler to run the code written for that event.
User	The person or team of persons who will be using the program when it is eventually handed over.
User documentation	Written document (online or paper-based) aimed at the user (i.e. not an IT professional) telling them how to install / use / deinstall the system / program in question.
User interface	The form where the user works with the program, usually putting data in or getting data out.
Variable	An area of electronic storage that has a name and type (as defined by the programmer) and will hold data at some point during program execution.

Design View



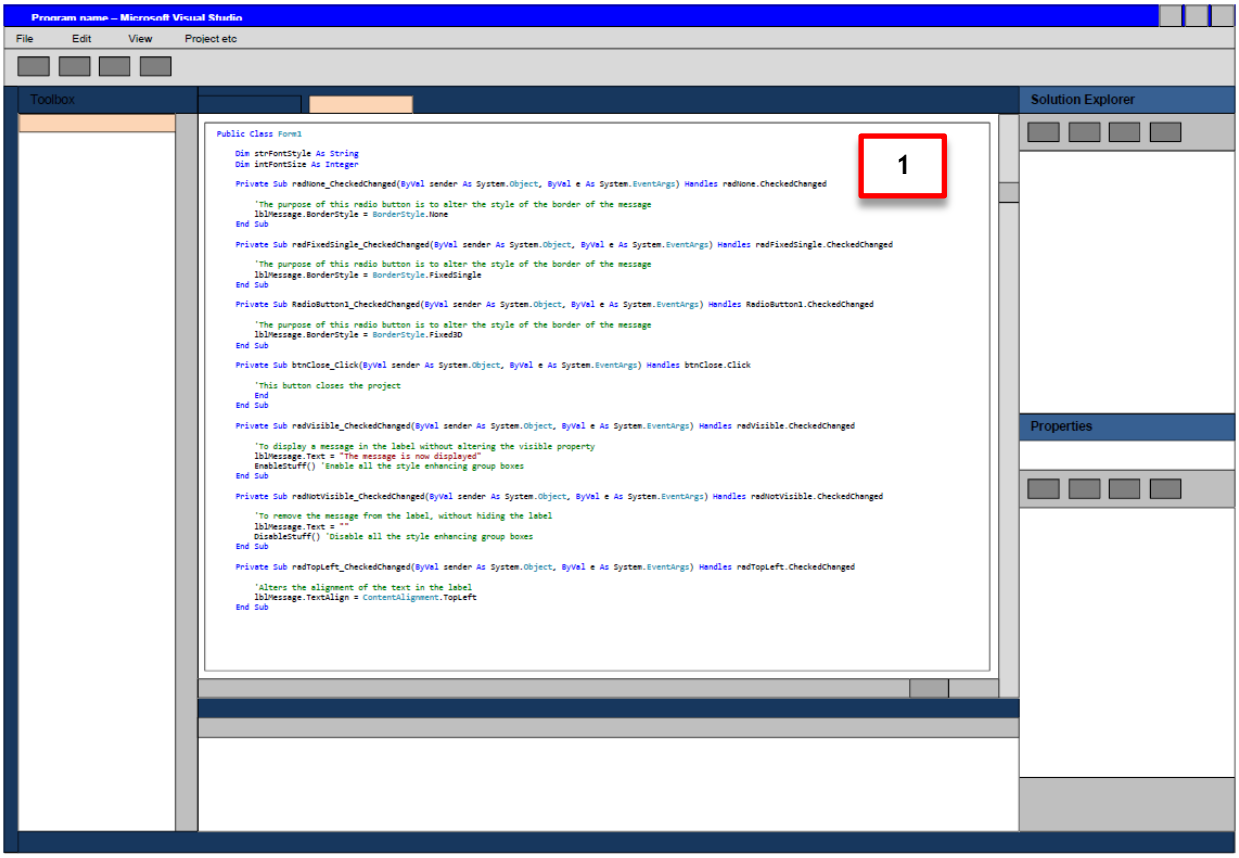
Key:

1. Menu bar
2. Menu icons (can be customised)
3. Toolbox (containing objects in design view)
4. Editing panel for active form
5. Minimise/Maximise/Close buttons for the software
6. Solution Explorer
7. Properties Window – default to "Alphabetical" as the display method (see arrow)
8. Output or Debug window



The illustration above is the usual layout of the Visual Basic Editor. Individual developers can tailor the Editor to meet their own needs but when you start out as a programmer it is best to stick to the layout above. If you "lose" one of the windows, try going to "View" on the menu bar (item 1 above) and click on the missing window's name there. If you find that the new window that you've just found opens up in the wrong place, right click on the blue bar at the top of that window and click "Dock".

Source Code View



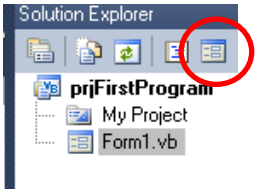
Key:

1. Code window

In order to access the code window to write the code for a specific object, just double-click on the object itself in the design view's editing panel (item 4 on the previous page). There are other ways of accessing the code window, but as an introduction that one is by far the easiest.

Notice that the Toolbox, Properties Window and Solution Explorer remain in the same place. They do, however, look slightly different in each view; the properties for an object aren't accessible from the code window, for example.

TIP:
If you open up a program and your form appears to have vanished, you're probably in the Source Code View of the editor. Go to the Solution Explorer and click on the name of the form you're wanting to see, then click on the "View Designer" button on the top line of the Solution Explorer.



Naming conventions for objects / controls

Lecturer: Jane Fletcher

Within the programming world it has become the custom to name objects / controls and variables using a set naming convention. This is so that it is possible to tell at a glance when reading code to what type of object or variable that code is processing / accessing at that time.

Most objects and variables have a prefix of three letters, each of which are written in lower case. The letters that follow should be descriptive of what the object or variable does, and each word in the name should start with an upper case letter and subsequent letters should be in lower case. No spaces are allowed in variable or object names. Numbers are allowed.

It is good practice to avoid leaving objects with their default names – this is sloppy practice – if that object is to be referred to within the code. If the object is NOT referred to within the code then it is acceptable to keep the default name. An example of this would be if a group of objects are clustered within a GroupBox object but that GroupBox itself is not referred to during coding, then it may retain its default name of GroupBox1 (or GroupBox2 etc).

The list below contains a list of some of the more commonly used controls, their prefixes and some example names. Notice that each of the example names follows the rules noted in this section. For what the controls actually do, see the section of this document that goes into detail for that control.

The blank lines at the bottom of the list can be used for you to make a note of any other controls that you use, together with the correct prefix and an example name.

Object	Prefix	Example name
Form	frm	frmSplashScreen
Button	btn	btnClose
Label	lbl	lblDisplayMessage
GroupBox	gb	gbUserInputSet
TextBox	txt	txtForename
RadioButton	rad	radItalicized
CheckBox	chk	chkBoldFont
ListBox	lst	lstFontChoice
MenuStrip	ms	msFormMainMenuStrip
DateTimePicker	dtp	dtpInvoiceDate
TabControl	tb	tbStarterMenu
Timer	tmr	tmrElapsedTime
PictureBox	pb	pbCompanyLogo

Commonly used controls - Form control

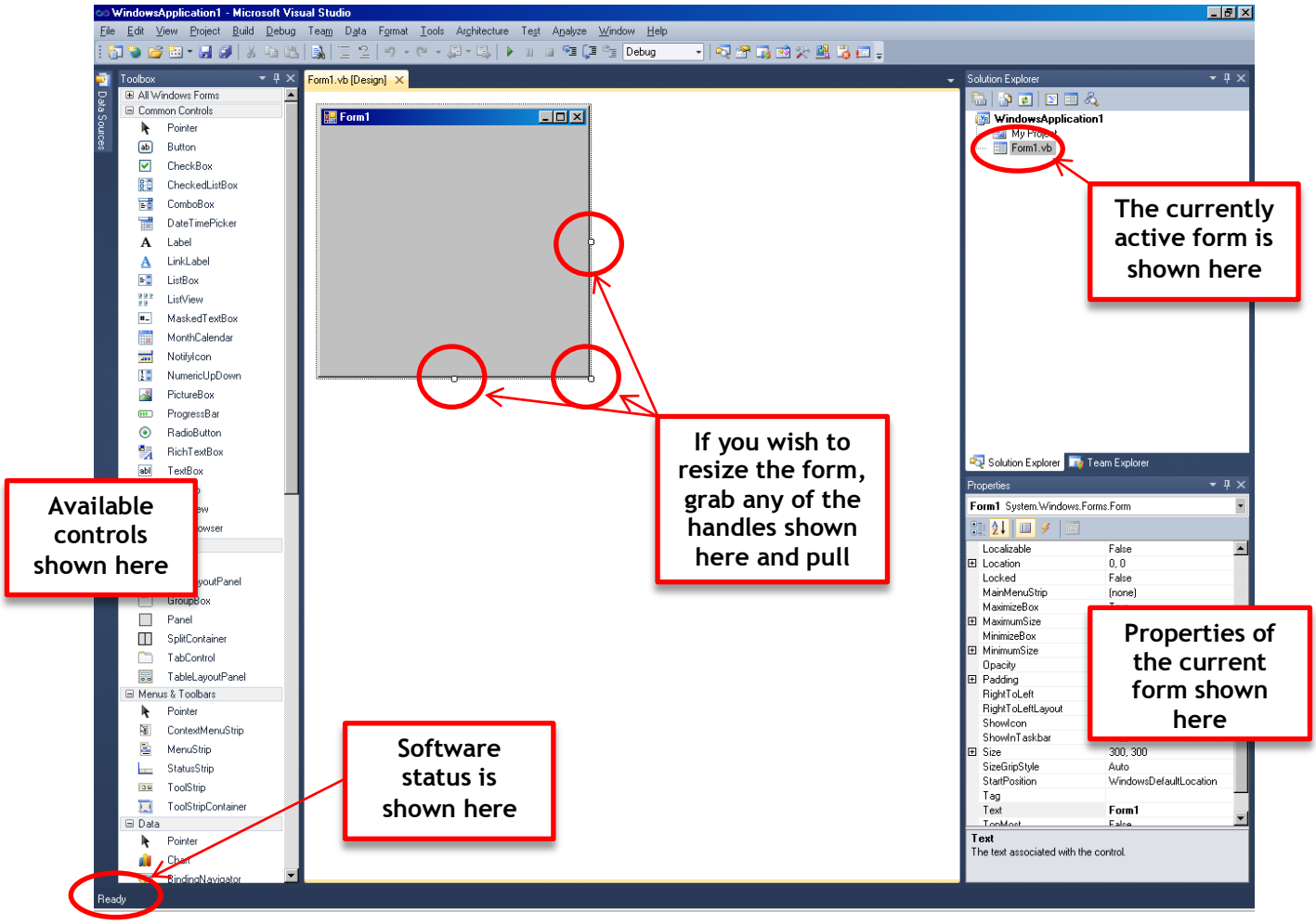
Lecturer: Jane Fletcher

In this section we will discuss the most commonly used objects used within this unit, and the properties that determine what it looks like and some of the ways in which it will behave.

Forms

One of the most commonly used objects within any visual program is the Form object. The Form is the object that holds all the others together, and is the one that presents the program "image" to the user. The list of commonly used properties over the page is by no means definitive, but is as the title suggests, a list of the properties most often changed within a form.

Below is a blank form in Designer view.



Commonly used controls – Form control

Lecturer: Jane Fletcher

Commonly used properties for the Form control

Name

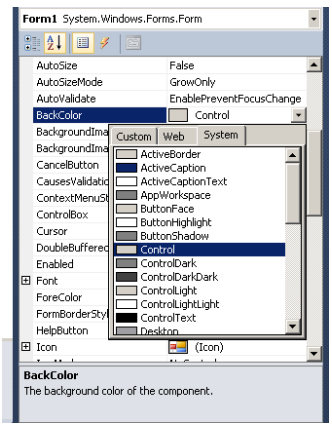
The Name property is what you want to call the form within the program. It is usually left as its default, which is Form1, Form2 and so on. If you wish to change the Name property, the prefix of the form name should always be "frm" (lower case) followed by a descriptive name with the first letter of the first and subsequent words being in upper case with the rest of the letters in lower case. Example names would be frmSplashScreen, frmLoginScreen and so on.

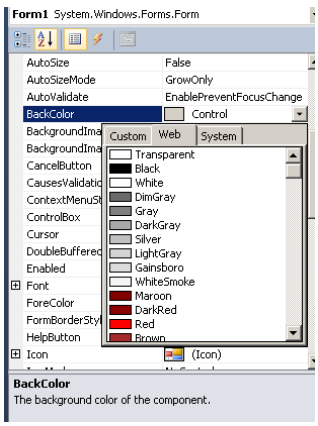
AcceptButton

The AcceptButton is the name of the control / object (usually a button) whose code will be executed if the user presses the Enter or Return key when the software is running.

BackColor

The colour of the background of the form. This will also determine the default background colour of all objects that are placed on the form (with exceptions). If you wish to change the background colour of an object on the form you must alter the BackColor property of that object individually. To change a background colour, click on the downward arrow to the right of the BackColor property in the Properties Window. From here there are several ways of changing the background colour. The first is to select a colour from the System palette displayed as default, shown in the first image below. A wider choice is available by clicking on the Web tab, as shown in the second image below. A further choice is available on the Custom tab, shown in the third image below.





ControlBox

If you set this to False then the form will not have the minimise, maximise and close buttons on the top blue status bar. If you leave it set to True (the default value) then they remain.

Commonly used controls – Form control

Lecturer: Jane Fletcher

Font The font style, settings and size of all text to appear on the form, with a default of MS Sans Serif, font size 8.25. Within the Font dialog box (which appears when you click on the ellipses to the right of the Font property in the Properties Window) the developer may select the type, the style and the size of the font. If you wish the font of an object on the form to be different, then set that object's Font property accordingly. The rule is that the font of the form is the font of all that form's objects, unless they're set differently.

ForeColor The colour of the writing on the form. This property determines the foreground colour of each of the objects on the form unless they're set individually. To change the foreground colour use the same method as to change the background colour of an object.

FormBorderStyle The appearance of the edges of the form. The choices are None, Fixed Single, Fixed3D, FixedDialog, Sizable, FixedToolWindow and SizableToolWindow. The default value is Sizeable. The best way to find out what the different styles look like is to have a go altering the FormBorderStyle property of a form and having a look at the effect.

MaximizeBox This property is set to either True or False depending on whether the programmer wants the user to be able to maximise the form. Normally it is good practice to take this option away since it makes the software look unprofessional and unbalanced unless the programmer centralises the main pane.

MinimizeBox This property is set to either True or False depending on whether the programmer wants the user to be able to minimise the form.

Size The actual dimensions of the form, usually defaulting to 300 by 300 (measurement unit default being pixels). The first figure of the two is the width, and the second is the height. It is possible to alter these during run time within code.

StartPosition This property's setting determines where the program will load on the user's screen when it is running. The default value of this property is WindowsDefaultLocation, which allows the program's form to load anywhere on the user's screen. It is far more professional to select CentreScreen for any form which is not either a parent or child form.

Text The words that appear on the blue bar of the program when it is running.

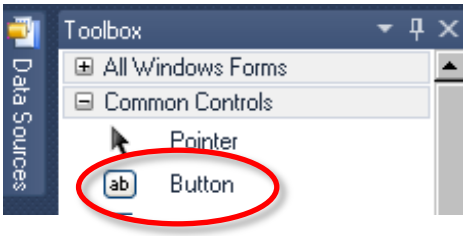
WindowState The way in which the form loads. The default value is Normal, but it is possible to make a form load in maximised or minimised state.

Commonly used controls – Button control

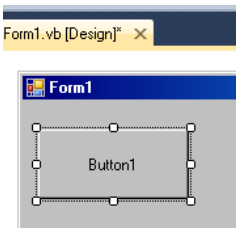
Lecturer: Jane Fletcher

Button

The button control is accessed from the Common Controls section of the Toolbox by clicking on the icon shown here. It is. the second one down after Pointer (this order is alphabetical). After the Form and the Pointer it is probably the most used control. It is used to prompt and enable the user to perform some processing on entered information.

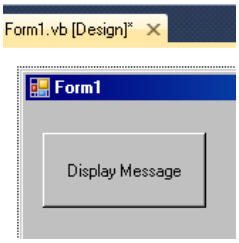


The way to get a button onto your form is to click on the button icon in the Toolbox (as above) and move across to the form with it. You will notice that the cursor changes from an arrow pointer to look like a plus sign with the button icon attached to it. When you're in the location where you want to place the button, press the left mouse button and drag the resulting rectangle shape to the size that you wish the button to be. You can move the button or resize it using the grab handles located around the outside of the button shape.

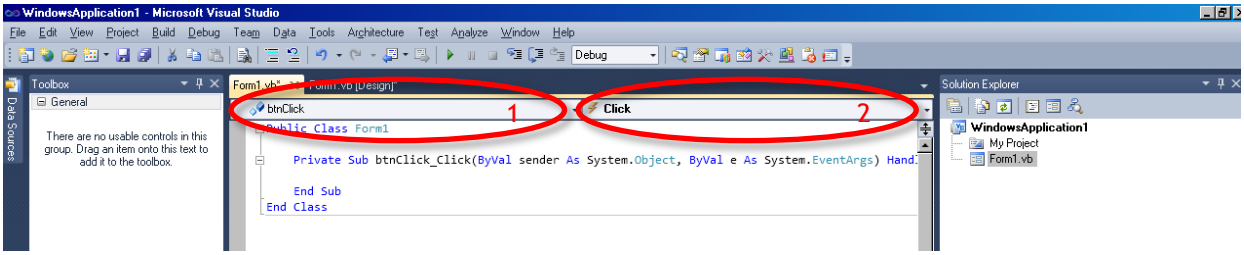


Notice that the text in the centre of the button contains the default name of this button object – the default name is Button1, with the next button used on the form having a default name of Button2 and so on. Notice that when you alter the name of the button object in the Properties Window, the text remains the same until you alter the Text property of the button.

In the example alongside you will see that the Text property of our button has been changed in the Properties Window to "Display Message". To do this, make sure that the button is selected (click once on it and notice that the grab handles will appear – this shows that the object has been selected) and then move to the Properties Window. Scroll to the Text property and delete "Button1". Type in "Display Message". Click back onto the Form object and notice that the text of the button has changed.



The default event of the Button object is the Click event. In order to get to the Source Code window for the button we've just created, double click anywhere on the button object on your form. The Click event for that button will be created, and you will be in the Source Code Window. You will see something like the image below, and your cursor will be flashing between the lines containing "Private Sub btnClick_Click" and "End Sub".



The Click event is, as stated above, the default event for the Button object. However, there are many other events that it is possible to access. To get to those, make sure that the button object's name appears in the Objects listbox (see "1" in the illustration above) and pull down the list of events in the Events listbox (see "2" in the illustration above).

Commonly used controls – Button control

Lecturer: Jane Fletcher

Commonly used properties for the Button control

Name The Name property is what you want to call the button within the program. If you wish to change the Name property, the prefix of the button name should always be "btn" (lower case) followed by a descriptive name with the first letter of the first and subsequent words being in upper case with the rest of the letters in lower case. Example names would be btnDisplayLabel, btnProcessInput and so on.

BackColor The colour of the background of the button. Unless you change this property the button will have the default background colour the same as that of the Form object upon which it sits. It is possible to change this property at design time as well as within the program's code. See the Form control for a deeper explanation as to how to change the BackColor property.

Enabled This property defaults to "True", which means that the button is useable within the form. If the property is set to "False", then the button will appear greyed out on the form at run time and the user will not be able to click on it. It is possible to change this property at design time as well as within the program's code.

Font Determines the size, style and type of font to be used in the Text property of the button. Clicking on the ellipses to the right of the Font property in the Properties Window generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the button. Unless you change this property the button will have the default foreground colour similar to the Form object.

Location Where the object will appear on the form, using the top left-hand corner of the form as a starting point. The location is defined using two figures (x and y) where x is the distance from the left-hand margin of the form and y is the distance from the top of the form. The default unit of measurement is the pixel (one centimetre is approximately 47 pixels).

Name The default prefix for a Button control is "btn".

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

Commonly used controls – Button control

Lecturer: Jane Fletcher

TabIndex

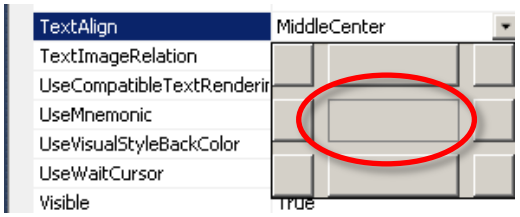
The VB .NET Editor gives each object placed on the form a TabIndex relative to when it was placed at design time; the first object put onto the form gets a TabIndex of 0, the second has a TabIndex of 1 and so on. The TabIndex determines the order in which control is passed on the form at run time when the user presses the Tab key. Ideally the tab order of a form should start where the cursor needs to be placed when the form loads and working its way up to the last field the user has to fill in on the form, in that order. It is pretty much guaranteed that the programmer will get the order of placement wrong in the first instance; if the tab order of a form is incorrect then the programmer can manually alter the TabIndex properties of all controls to ensure that the lowest number (i.e. 0) is given to the first place the cursor should go, working up to the final place where the user needs to be to enter information onto the form with this object having the highest number in its TabIndex property. Any objects that don't require user input (like a prompt, for instance) can be given a TabIndex of 100. It doesn't matter to VB .NET if higher numbered TabIndexes are duplicated.

Text

The words that appear in the centre of the button. Be succinct, ensure that all of the text is visible and never ever use bad language!

TextAlign

Text within a button can be displayed using any form of alignment, selected by using the TextAlign property in the Properties Window. The default value is MiddleCentre but to change it click on the box representing the position where you want the text to appear. The selected position has a flat appearance in the TextAlign box, as shown above.



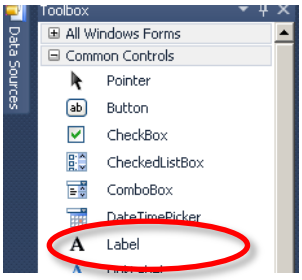
Visible

This property can be set to equal "True" or "False" both at design time and within code. The developer may wish to have a button hidden until certain conditions are met, in which case the Visible property is more appropriate than the Enabled property.

Commonly used controls – Label control

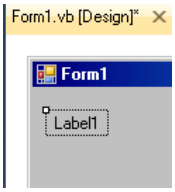
Lecturer: Jane Fletcher

Label

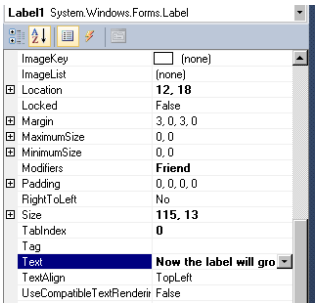


The Label control is accessed from the Common Controls section of the Toolbox by clicking on the icon shown here. The Common Controls are listed in alphabetical order.

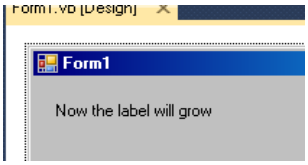
Labels are used either as prompts, or to display information to the user. They do not allow user entry at all. The default event of a Label object is the Click event, but as a general rule labels are not used for anything other than displaying information. It isn't normal, for example, to get a user to click on a label when they wish to get the software to perform some sort of processing.

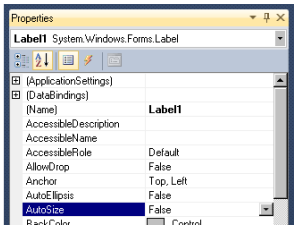


In order to get a label onto your form, click on the Label control in the Toolbox, move across to the form and where you wish the label to be, drop it. If you want a larger label that will enable text to wrap, and one that will always be a certain size regardless of its contents, then drag the cursor to be the size that you require. The label will shrink back to enable just the name of the label (default name being Label1, then Label2 etc).



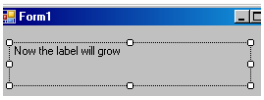
When you enter text into the label (via the Text property in the Properties Window), it will stretch to accommodate the new words.





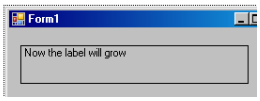
In order to allow the label to be the size you require, you must set the AutoSize property to be "False" in the Properties Window.

The result of doing this is:

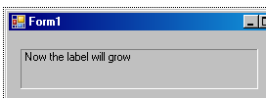


Now you can alter the BorderStyle property in the Properties Window to find a layout that you like the look of. The image above shows that the label has a BorderStyle of "None". The borders of the label are shown with a dotted line – the one above also shows that the label is selected (notice the grab handles around the edge) because otherwise it isn't possible to show where the outline of the label is.

This label has a BorderStyle of FixedSingle:



This label has a BorderStyle of Fixed3D:

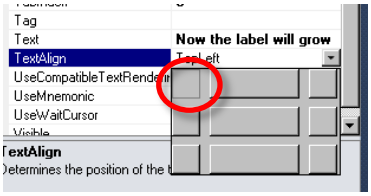


Notice that the text contained in the labels in each of the illustrations above is located in the top left-hand corner of the label object. While this can be appropriate it doesn't look particularly visually appealing if the purpose of the label is to display some form of output.

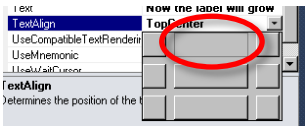
Commonly used controls – Label control

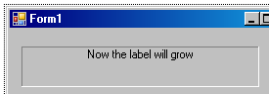
Lecturer: Jane Fletcher

It is possible to alter the location of the text within the label by use of the label's TextAlign property in the Properties Window for that control. The default value is TopLeft. By clicking on the grid of nine boxes (see illustration here) you can move the location of where the text of the label will occur. The current location is shown as the button that appears to have been pressed in.

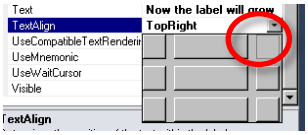


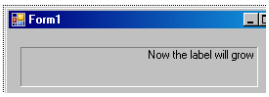
TopCenter





TopRight





And so on.

Commonly used controls – Label control

Lecturer: Jane Fletcher

Commonly used properties for the Label control

AutoSize This property has a default value of "True". A label that has an AutoSize set to true will grow to fit the text that the developer puts in its Text property. However should the label contain a large amount of text it may possibly extend beyond the right-hand edge of the form, since it isn't possible to wrap text in a label with AutoSize set to True. There are no grab handles on an auto-sized label so it isn't possible to stretch it to fit the text it is to hold. Setting this property to "False" allows the programmer to set the size, border style and alignment of the text it is to contain.

BackColor Sets the background colour of the label. The default background colour is determined by that of the form, and it can be changed using the same method as that applied when altering the form's background colour.

BorderStyle The default BorderStyle is "None", which means that the label will have no edges. Other options are "FixedSingle" which would give the label a black line drawn around its edges, or "Fixed3D" which makes the edges of the label appear in relief, giving a reverse embossed effect. If the software needs to draw attention to some information displayed within a label setting a different border style can draw the user's eye to it.

Enabled It isn't usual to enable a label since it doesn't normally have any coded event attached to it. However if there is an event that will occur then setting the Enabled property to "False" in either the Properties Window at design time or during run time will cause the label to appear to be greyed out and will disable any event handler attached to it. Setting the Enabled property to "True" will turn it back on again.

Font Determines the size, style and type of font to be used in the Text property of the label. Clicking on the ellipses to the right of the Font property in the Properties Window generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the label. Unless you change this property the label will have the default foreground colour similar to the Form object.

Location Where the object will appear on the form, using the top left-hand corner of the form as a starting point. The location is defined using two figures (x and y) where x is the distance from the left-hand margin of the form and y is the distance from the top of the form. The default unit of measurement is the pixel (one centimetre is approximately 47 pixels).

Name The default prefix for the Label object is "lbl".

Commonly used controls – Label control

Lecturer: Jane Fletcher

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

TabIndex Unless the label is to receive control during run time then the programmer can manually alter the TabIndex to be a high number, using the Properties Window at design time. See the explanation of the purpose of the TabIndex in the Button object.

Text The words or numbers that are displayed in the label. The label can display anything – there is no restriction in terms of what characters can be used since the Text property of a label is classed as a literal.

TextAlign Text within a label can be displayed using any form of alignment, selected by using the TextAlign property in the Properties Window. The default value for the Label object is TopLeft but to change it click on the box representing the position where you want the text to appear. The selected position has a flat appearance in the TextAlign box, as shown above.

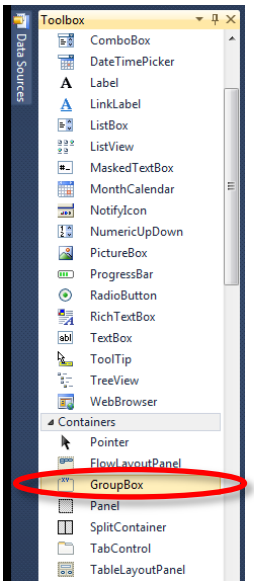
Visible This property can be set to equal “True” or “False” both at design time and within code. The developer may wish to have a label hidden until certain conditions are met, in which case the Visible property is more appropriate than the Enabled property.

Commonly used controls – GroupBox control

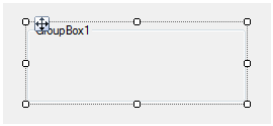
Lecturer: Jane Fletcher

GroupBox

The GroupBox control, as its name suggests, is used to group together other controls that need to either work together as a team (such as radio buttons) or for visual effect (i.e. controls that are to work together to perform a specific purpose) to either look good or to show the user that these controls are part of the same groups of input or output.

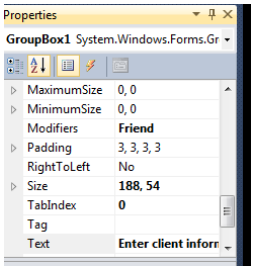


The GroupBox control can be found in the Toolbox under the "Containers" menu. In order to get a GroupBox onto your form, select the GroupBox object in the Toolbox by clicking on it, and move across to the form. When you get the location on the form where you would like to place the top left-hand corner of the GroupBox press the left mouse button to drop the box, then, keeping the left mouse button pressed, drag the GroupBox until it is the size and shape that you need and then release the mouse button.

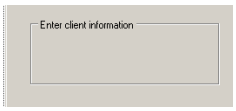


Notice that the control has an inbuilt text feature which by default is the name of the GroupBox. The default name for the GroupBox control is GroupBox1, followed by GroupBox2 and so on. In order to change the text ensure that the GroupBox is the active object and then alter its Text property in the Properties Window.

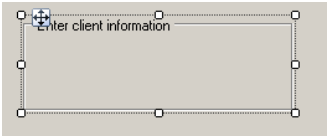
Change Text property:



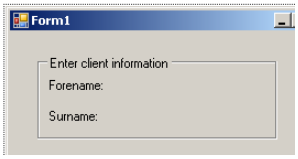
Display after hitting the Return key:



When using a GroupBox it is important to remember that when you're actually attempting to place other objects within the GroupBox itself, you must have selected the GroupBox so that it is the active object. You can tell that it is by the fact that grab handles are visible around its edges. Incidentally you can use these grab handles to resize the GroupBox in the same way as you can with other controls.



To the right is the GroupBox with two labels contained within it.



If you do not ensure that you have selected the GroupBox prior to the addition of other controls, you may find that these new controls don't behave as if they are part of the GroupBox. A good way of testing to see if the controls think they "belong" is to move the GroupBox – if the controls within it move too, then they are part of that Group. To move the GroupBox, select it and place the cursor in its top left-hand corner. Notice that a small icon with north, south, east and west pointers appear. Hold the left mouse button down over this icon and move the GroupBox to where you want it to be. Release the mouse button when you're there.

Commonly used controls – GroupBox control

Lecturer: Jane Fletcher

Commonly used properties of the GroupBox control

BackColor Sets the background colour of the GroupBox. The default background colour is determined by that of the form, and it can be changed using the same method as that applied when altering the form's background colour.

Font Determines the size, style and type of font to be used in the Text property of the GroupBox. Clicking on the ellipses to the right of the Font property in the Properties Window generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the GroupBox. Unless you change this property the label will have the default foreground colour similar to the Form object.

Name The default prefix for a GroupBox control that is to be referred to in the code is "gb".

Size Given as an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

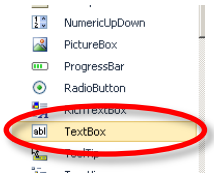
Text The words or numbers that are displayed in the Text property of the GroupBox. This text will be displayed in the top left-hand corner of the GroupBox.

Visible Set to either "True" or "False" – determines whether the GroupBox (and all its contents) are visible or not. Can be changed at either design time or run time.

Commonly used controls – TextBox control

Lecturer: Jane Fletcher

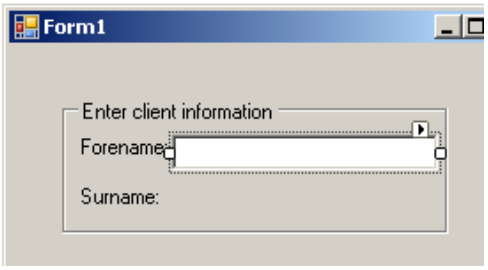
TextBox



The TextBox control is accessed from the Common Controls section of the Toolbox by clicking on the icon shown here. The Common Controls are listed in alphabetical order.

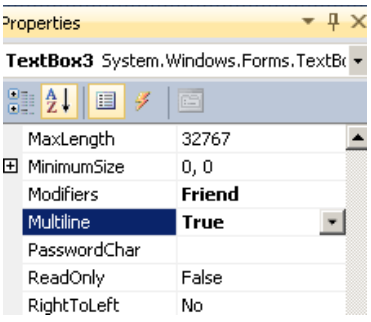
TextBoxes are used to gain information from the user; unlike labels that do not allow user entry at all, that is the sole purpose of the TextBox; as a general rule TextBoxes are not used for anything other than gaining information from the user. It isn't normal, for example, to get a user to click on a TextBox when they wish to get the software to perform some sort of processing.

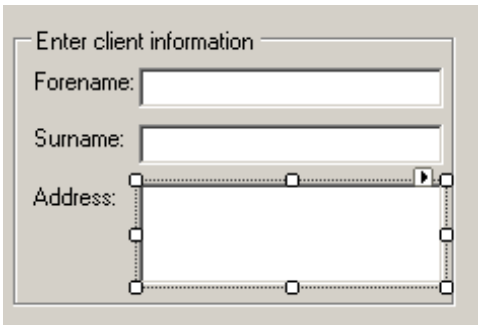
The way to add a TextBox to a form is to click on the TextBox icon in the Toolbox, move across to the Form object and press the left mouse button at the place on the form where the top left-hand corner of the TextBox is to be situated. Drag across to describe the width of the TextBox that you want. Note that you can't actually alter the height of the TextBox at this stage, since by default TextBoxes don't allow a multiline user entry and the height of the box is determined by its Font property. The image to the right shows an active TextBox that has been placed in a GroupBox and alongside a Label (used as a prompt to the user to show in this case that the TextBox is to be used for inputting information on the client's forename.



The default event of a TextBox object is the TextChanged event. The TextChanged event is triggered every time the program detects that there has been some form of change to the TextBox – this occurs every time there is a change; in other words, for every character typed into the TextBox by the user. In view of the fact that it isn't usual for the developer to want to check one character at a time what the user has entered into a TextBox, the TextChanged event isn't normally the one that is selected to handle any validation necessary – this isn't always the case, but in the vast majority of cases it is. It is more usual to have a button associated with a group of TextBoxes that will check user entry once it is complete. The button will run on the Click event.

Should you wish to have a TextBox that allows more than one line of user entry then it is necessary to set the Multiline property to "True" (it defaults to "False"). In that way the TextBox can be stretched downwards to accommodate the extra text.





In the example here, TextBox3 has its Multiline property set to "True", thus allowing the developer to have a longer TextBox. This is appropriate for a field that requires more data to be input by the user.

Commonly used controls – TextBox control

Lecturer: Jane Fletcher

Commonly used properties of the TextBox control

BackColor Sets the background colour of the TextBox. The default background colour is determined by that of the form, and it can be changed using the same method as that applied when altering the form's background colour.

Enabled Default value of "True", meaning that the TextBox can have user entry when it is the active control on the form. A TextBox that is not enabled (Enabled property of "False") will not allow the user to click in the TextBox or the developer to set the focus of the program to the TextBox.

Font Determines the size, style and type of font to be used in the Text property of the TextBox. Clicking on the ellipses to the right of the Font property in the Properties Window generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the TextBox. Unless you change this property the label will have the default foreground colour similar to the Form object.

Location Where the object will appear on the form, using the top left-hand corner of the form as a starting point. The location is defined using two figures (x and y) where x is the distance from the left-hand margin of the form and y is the distance from the top of the form. The default unit of measurement is the pixel (one centimetre is approximately 47 pixels).

Multiline Defaults to "False". A TextBox with a Multiline property of "True" will allow the user to type in multiple lines of text into the box.

Name The default prefix for the TextBox object is "txt".

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel. Unless Multiline is set to "True" the height of the box cannot be altered – the height is determined by the size of the box's Font property.

TabIndex Unless the label is to receive control during run time then the programmer can manually alter the TabIndex to be a high number, using the Properties Window at design time. See the explanation of the purpose of the TabIndex in the Button object.

Text The writing that appears in the box. Normally it will start out blank although default text can be put in the box's Text property at run time by the software.

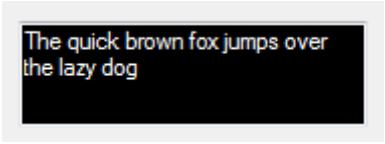
TextAlign Where the text will appear in the TextBox. The default value is "Left", but "Centre" and "Right" are also available.

Commonly used controls – TextBox control

Lecturer: Jane Fletcher

Visible Set to either "True" or "False" – determines whether the TextBox is visible or not. Can be changed at either design time or run time.

WordWrap Default value is "True". If the Multiline property of the TextBox is also set to True then text will wrap around onto the next line if the user's entry is too long to fit onto one line of the text box. In the illustration below the first TextBox has its WordWrap set to "False" and Multiline as "True", whereas in the second illustration both WordWrap and Multiline are set to "True". Notice the difference.

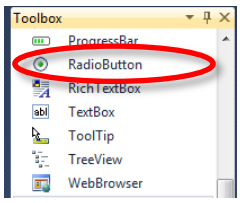


Commonly used controls – RadioButton control

Lecturer: Jane Fletcher

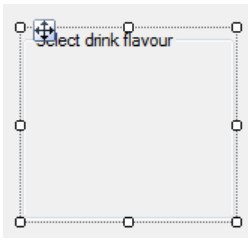
RadioButton

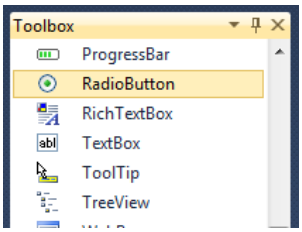
The RadioButton control is accessed via the Common Controls section of the Toolbox by clicking on the icon shown here.



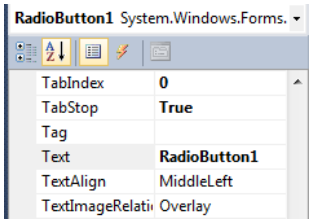
RadioButtons are designed to work in "teams" that sit together within a GroupBox control. RadioButtons present the user with choices; within a team of RadioButtons only one can be selected at any one time. An example of when a team of RadioButtons might be used would be to select a flavour for a bottle of drink – it would be either orange or lemon or cola but it can't be all three, or two of the three. (I'm sure it could be but who would want to try the result?)

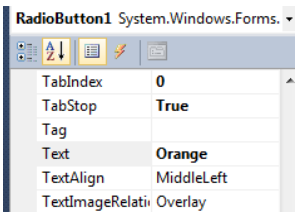
The way to get a RadioButton onto a form is to first of all place a GroupBox control for them to sit in. After that, make sure that the GroupBox is the active control (i.e. there are visible grab handles around its edges, as in the first illustration below) and then click on the RadioButton icon in the Toolbox (second illustration). Move across to the GroupBox and then press the left mouse button when you reach the location of where you want the RadioButton to sit (third illustration). Naming convention gives RadioButtons a prefix of "rad".





RadioButtons have their own associated label (Text property) used to display the nature of this choice to the user. In the example below the first illustration the default value in the Text property is the name of the object, which in the case of RadioButtons is RadioButton1, RadioButton2 and so on. The second shows the Text property having been changed to "Orange" and the third shows the result on the RadioButton itself.

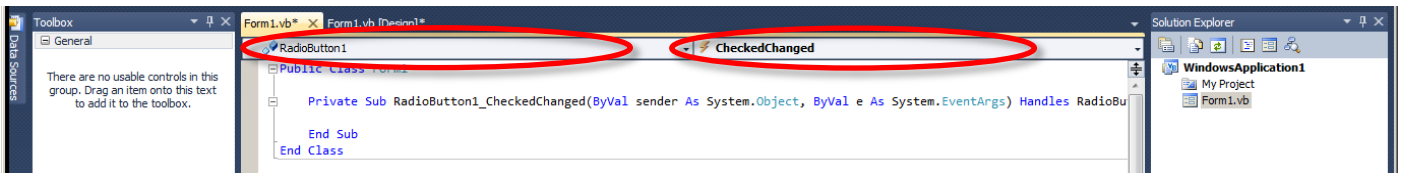




The default event for a RadioButton is the CheckedChanged event. This is activated when the status of the RadioButton is altered – going from either selected (with a green dot in the middle of the RadioButton) to deselected (no green dot in the middle) or vice versa. However, there are many more other events associated with the RadioButton control which can be explored by ensuring that the RadioButton is selected in the Objects ListBox and pulling down the list in the Events ListBox in the Source Code Editor view, as shown over the page.

Commonly used controls – RadioButton control

Lecturer: Jane Fletcher



Commonly used Properties of the RadioButton control

BackColor Sets the background colour of the RadioButton. The default background colour is determined by that of the GroupBox in which it sits, but it can be changed using the same method as that applied when altering the form's background colour. Bear in mind that when you change the BackColor of a RadioButton to be different to that of the GroupBox it "belongs to", the label associated with the RadioButton is what will change, not the actual "dot" of the radio button itself.

Checked Default value is "False", meaning that the RadioButton is not selected. When the user clicks on a deselected RadioButton its Checked property will change to "True" and vice versa.

Font Determines the size, style and type of font to be used in the Text property of the RadioButton. Its default value is to take the Font property of the GroupBox in which it sits. Clicking on the ellipses to the right of the Font property in the Properties Window for the RadioButton generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the RadioButton's associated label. Unless you change this property the label will have the default foreground colour similar to its owning GroupBox object.

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

TabIndex Unless the label is to receive control during run time then the programmer can manually alter the TabIndex to be a high number, using the Properties Window at design time.

Text The writing that appears in the RadioButton's label. Normally it will start out with the default name of the RadioButton.

TextAlign Where the text will appear in the RadioButton's label. The default value is "MiddleLeft", but each of the choices available to the Label object is available here.

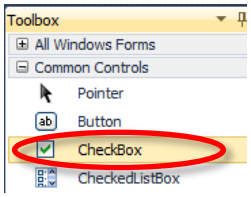
Visible Set to either "True" or "False" – determines whether the RadioButton and its associated label are visible or not. Can be changed at either design time or run time.

Commonly used controls - CheckBox control

Lecturer: Jane Fletcher

CheckBox

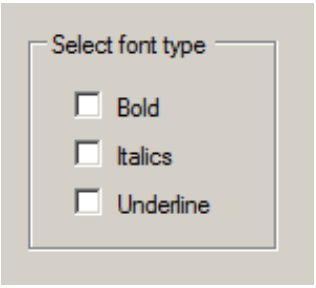
The CheckBox control is accessed via the Common Controls section of the Toolbox by clicking on the icon shown here.



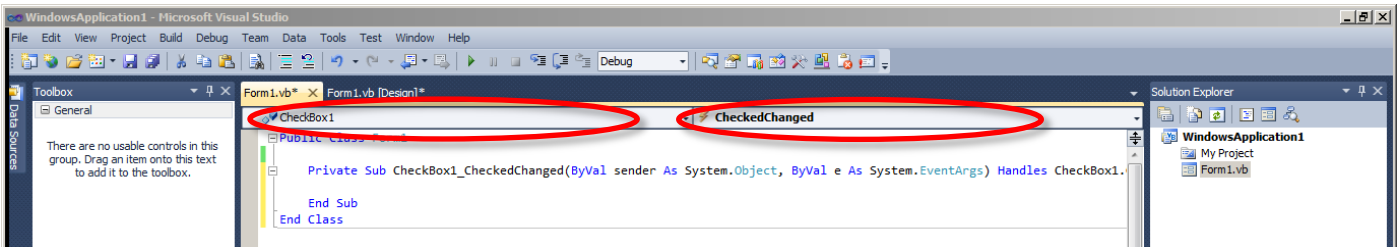
Like RadioButtons, CheckBoxes are generally grouped together in a GroupBox, but are not mutually exclusive so it isn't so important to get them working together as a team as it is with RadioButtons. It is quite possible to put CheckBoxes into the same GroupBox as RadioButtons since they won't interfere with each other in terms of only one being allowed to be active.

The way to get a button onto your form is to click on the CheckBox icon in the Toolbox (as above) and move across to the form with it. You will notice that the cursor changes at the point of entering the vicinity of the form from a pointer to a plus sign with a small image of a CheckBox beneath it. When you reach the area of the form where you want to place the CheckBox, press the left mouse button and drag across to the right, ensuring you allow enough room for the associated label. If you are putting the CheckBox inside a GroupBox then make sure that the GroupBox is active before clicking in the Toolbox to get the CheckBox.

In the illustraton alongside, the form contains a GroupBox and has had three CheckBox objects added. Their associated Text properties (the labels alongside the check boxes) have been changed to "Bold", "Italics" and "Underline". For this particular example, font style may be bold (or not), italicised (or not) and underlined (or not). It might be that none of these items are selected, or all of them, or any combination of them. In this way the CheckBox object differs from the RadioButton object, where only one of a set of RadioButtons can be true.



The default event for a CheckBox is the CheckedChanged event. This is activated when the status of the CheckBox is altered - going from either ticked (checked) to not ticked (no check in the box) or vice versa. However, there are many more other events associated with the CheckBox control which can be explored by ensuring that the CheckBox is selected in the Objects ListBox and pulling down the list in the Events ListBox in the Source Code Editor view, as shown below.



Commonly used controls – CheckBox control

Lecturer: Jane Fletcher

Commonly used Properties of the CheckBox control

BackColor Sets the background colour of the CheckBox. The default background colour is determined by that of the GroupBox in which it sits, but it can be changed using the same method as that applied when altering the form's background colour. Bear in mind that when you change the BackColor of a CheckBox to be different to that of the GroupBox it "belongs to", the label associated with the CheckBox is what will change, not the actual "tick" of the check box itself.

Checked Default value is "False", meaning that the CheckBox is not selected. When the user clicks on a deselected CheckBox its Checked property will change to "True" and vice versa.

Font Determines the size, style and type of font to be used in the Text property of the CheckBox. Its default value is to take the Font property of the GroupBox in which it sits. Clicking on the ellipses to the right of the Font property in the Properties Window for the CheckBox generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the CheckBox's associated label. Unless you change this property the label will have the default foreground colour similar to its owning GroupBox object.

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

TabIndex Unless the label is to receive control during run time then the programmer can manually alter the TabIndex to be a high number, using the Properties Window at design time.

Text The writing that appears in the CheckBox's label. Normally it will start out with the default name of the CheckBox.

TextAlign Where the text will appear in the CheckBox's label. The default value is "MiddleLeft", but each of the choices available to the Label object is available here.

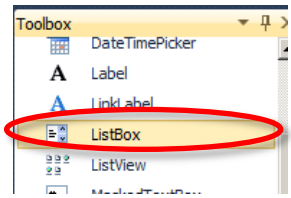
Visible Set to either "True" or "False" – determines whether the CheckBox and its associated label are visible or not. Can be changed at either design time or run time.

Commonly used controls – ListBox/ComboBox controls

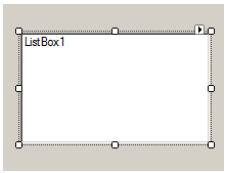
Lecturer: Jane Fletcher

ListBox

The ListBox control is accessed via the Common Controls section of the Toolbox by clicking on the icon shown here.



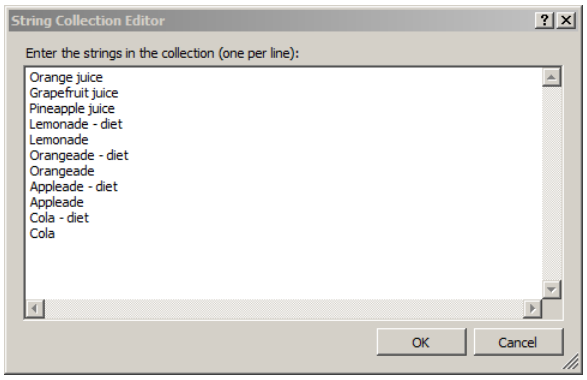
To place a ListBox onto your form, click on the ListBox icon in the Toolbox and move across to the form with it. When you are at the location on the form where you wish the top left-hand corner of the ListBox to be, left-click the mouse and drag across and down to describe your chosen size. Notice that the active ListBox has grab handles in the same way that other controls have.

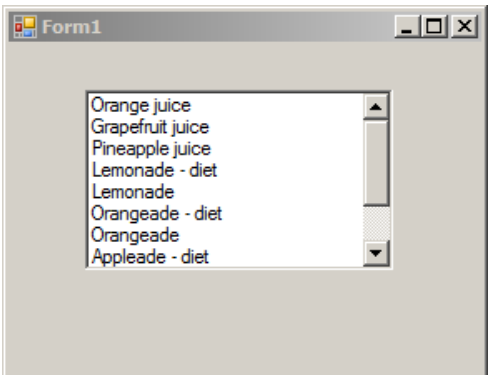


Notice that the text in the ListBox is the default name of the object – starting at ListBox1 and so on. You cannot type information directly into a ListBox; that has to be done via the Items property in the Properties Window or at run time by the program code.

> ***What is the difference between a ListBox and a ComboBox?***
> *The ComboBox object allows the user to type information into it at run time, and has a drop-down list of items contained within it. When the user makes a selection from the ComboBox's list of items, the ComboBox resumes its single-line shape and displays the user's choice. A ListBox on the other hand can be stretched downwards to display the entire list of data contained within it (at design time), or if that is too much then it will give a vertical scroll bar to the left-hand side of the list. The user cannot type information into the ListBox control. When the user makes a choice from the ListBox's list of items then that line of the list is highlighted but other items on the list still remain visible. An example of a ComboBox in action is the URL bar in a web browser. An example of a ListBox is the list shown in the Favourites window. You can type into one but not the other.*

In the example below you will see that the "Items" dialog box has had several items added to it (by clicking on the ellipses to the right of the Items property). You cannot type directly into the Items property itself – that displays the word "Collection". The second illustration shows the same ListBox at run time – this is what the user sees when using the program.





Notice in the second illustration the vertical scroll bar that the program has automatically generated due to the fact that there are too many items in the list to be displayed in the space available. The developer doesn't have to undertake any extra coding to make this happen.

Commonly used controls – ListBox/ComboBox controls

Lecturer: Jane Fletcher

Commonly used Properties of the ListBox control

BackColor Sets the background colour of the ListBox. The default background colour is determined by that of the form in which it sits, but it can be changed using the same method as that applied when altering the form's background colour.

BorderStyle The default BorderStyle is "Fixed3D", which means that the ListBox will appear in relief giving a reversed embossed effect. Other options are "FixedSingle" which would give a black line drawn around the edges of the ListBox, or "None".

Font Determines the size, style and type of font to be used in the Items property of the ListBox. Its default value is to take the Font property of form. Clicking on the ellipses to the right of the Font property in the Properties Window for the ListBox generates the Font Dialog Box which allows the developer to select his or her choice of font/style/size.

ForeColor The colour of the text within the ListBox's Items collection. Unless you change this property the label will have the default foreground colour similar to the Form object.

Size Again an (x, y) figure where x is the width of the object and y is the height of the object. The unit of measurement is the pixel.

Sorted The default value for this property is "False". This means that the Items collection will appear in the order in which they were written in the ListBox. If the property is changed to "True" then the Items will display in alphabetical order, with any item beginning with a number will appear before any item beginning with a letter.

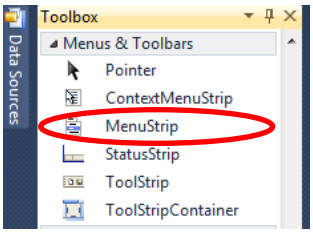
TabIndex Unless the label is to receive control during run time then the programmer can manually alter the TabIndex to be a high number, using the Properties Window at design time.

Visible Set to either "True" or "False" – determines whether the ListBox is visible or not. Can be changed at either design time or run time.

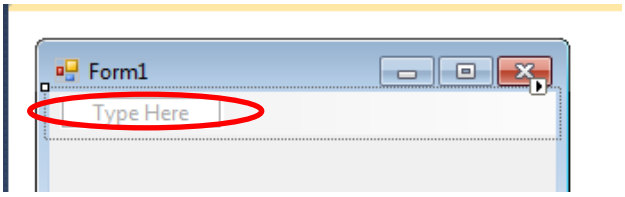
Commonly used controls - MenuStrip control

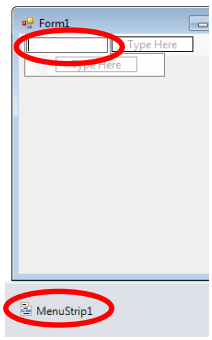
Lecturer: Jane Fletcher

MenuStrip

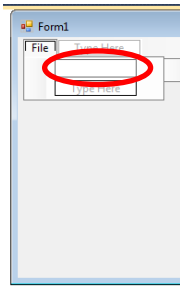


The MenuStrip control is accessed via the Menus & Toolbars collection in the Toolbox by clicking on the icon shown here. Once you have clicked on the MenuStrip object, move across to the form and left click anywhere on the Form object. The MenuStrip will appear at the top of the Form object, as shown here. You will also be able to see the MenuStrip icon in the lower grey pane of the Form object in a similar position to where the Timer object would go (see the illustration to the right here).

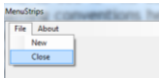




Notice the wording 'Type Here', greyed out in the top left-hand corner of the MenuStrip object. When you click on the words 'Type Here' a text box will open up and allow you to type the words that you would like to appear as your first item in the menu. Type it as you would like to see it - no abbreviations and do not employ any VB .NET naming conventions here. Type the word 'File' and then click in the lower box beneath the word 'File'. You'll be given the opportunity to enter an item within the File menu, such as 'Open', 'Save', 'Help' and 'Close'. Keep your menus as close to the 'normal' menus that users are accustomed to as possible because as we've already established, users hate change. When you've completed the 'File' menu, press the Tab key to get to the next menu on the top level of the MenuStrip to enter your next items.



Here is a form with a completed MenuStrip.



When Visual Basic .NET creates a MenuStrip within a program, it will automatically name each of the items you place within it. In the example above, we have 'FileToolStripMenuItem', 'NewToolStripMenuItem' and 'CloseToolStripMenuItem'. Since these names are clear and readily understood there is no need to change the defaults.

It is possible to change the direction of the text within the MenuStrip by altering the TextDirection property, and for the items within the menu to be shown right-aligned by altering the RightToLeft property to Yes.

To change the colours used within the MenuStrip object, change the BackColor as usual in the Properties window to each individual item in the sub-menus as well as at the top level. However ForeColor will have to be altered in the code, like this:

```
MenuStrip1.ForeColor = Color.Red
```

However, this will only alter the TOP layer of the MenuStrip so any lower levels (like New and Close in our example) will have to be altered individually:

```
NewToolStripMenuItem.ForeColor = Color.Red
CloseToolStripMenuItem.ForeColor = Color.Red
```

Commonly used controls – MenuStrip control

Lecturer: Jane Fletcher

Once you have a MenuStrip object on your form, it is extremely easy to make it work: all you have to do is within the design view, double-click on the lowest level of the menu to see the code view for that object's Click event, which is the default event of the MenuStrip. The difference between the MenuStrip object and many other objects is that the MenuStrip can have many Click events, one per lowest level item. By lowest level, in this context we mean items that do not have items of their own. In the example above, the 'File' menu item owns 'New' and 'Close', so that when 'File' is clicked, the event opens the sub-menu that includes 'New' and 'Close' without any coding being required from the developer. 'New' and 'Close' don't have any sub-menus, so they will have Click events of their own available to the developer to implement by double-clicking on them in design view.

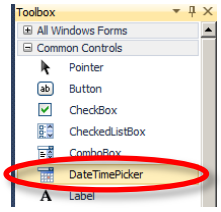
Below is the coding for the 'Close' event of the example above.

```
Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles CloseToolStripMenuItem.Click

    'Close the current project
    End
End Sub
```

Note that the underscore character included in the 'Private Sub' statement indicates that the statement is continued on the line below. The underscore must be preceded by a space, otherwise it will not be acknowledged and a compiler error will be generated.

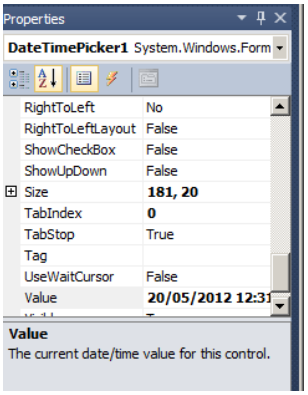
Items on the MenuStrip control can be enabled or disabled at both design and run time depending on logical need. Items that have their Enabled property set to False will appear greyed out at run time.

DateTimePicker

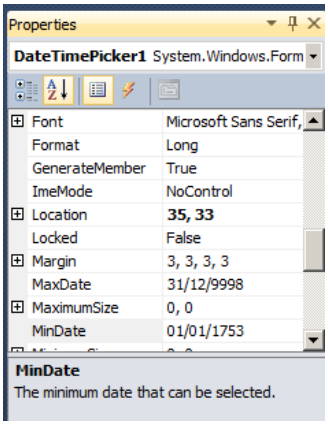


The DateTimePicker control is accessed via the Common Controls section of the Toolbox by clicking on the icon shown here.

To place a DateTimePicker on your form, click on the DateTimePicker icon in the Toolbox and move across to the form. When you're at the place where you want the top left-hand corner of the control to be, left-click the mouse button and drag across. Release the mouse button. The resulting DateTimePicker will look very similar to a ComboBox control in that it occupies just one line rather than taking the form of a square box like a ListBox would. Notice in the right side of the control there is a downward arrow which does nothing when you're in design mode but during run time this arrowhead will allow the user to view a full month calendar because it will drop down a display which will then rewind once the user has made a change by clicking on a date, or by clicking away from the DateTimePicker to anywhere else on the form. This section of this document was created on May 16th 2012, thus showing that the default value of the DateTimePicker is the current date. It is possible to change the default date by altering the Value property in the Properties Window of the control, as shown in the illustration to the right. Notice that the date has been changed to 20/05/2012 in the Value property, and that there is a time aspect to the DateTimePicker's value in addition to the date aspect. It is important to bear this information in mind when writing code because otherwise unexpected results can occur! See later on in this document for a further explanation of this. You will see that the contents of the Value property has been reflected into the starting display of the DateTimePicker.



DateTimePickers are the most efficient way of allowing users to enter any date information since it has inbuilt validation and the developer can stipulate the ranges of dates that can be selected, either by hard-coding dates into the MinDate and MaxDate properties or by coding so that minimum and maximum dates are worked out by comparison to another date (for example, today's date or somebody's birth date). Notice in the illustration to the right that the MinDate and MaxDate properties have a massive range as default which may not be what you want within your software, so remember to change them.

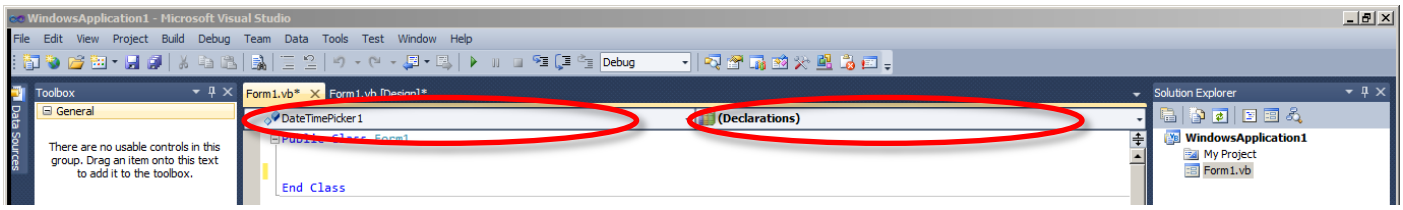


It is important to remember when using a DateTimePicker that the Value property contains both a date aspect as well as a time aspect (the key is in the control's name, really!) but because when at run time the control displays only a date choice, the time aspect is often overlooked. If you're trying to work out the difference between two dates for example, the time factor stored in a date variable taken from the Value property of a DateTimePicker will have an associated time and so if the first date to be used in the equation is in the afternoon and the second date in the morning, then you won't get a complete day's difference so the answer you have may be unexpected. The answer you get isn't in rounded-up full days so a difference of 23 hours and 59 minutes is 0 days!

Commonly used controls – DateTimePicker control

Lecturer: Jane Fletcher

The default event for the DateTimePicker control is the ValueChanged event, which occurs once the user has actively altered the selected date on at run time. There are other events available which can be explored in the Source Code view of the Editor by clicking on the DateTimePicker object's name in the left-hand ComboBox and opening up the list of available events in the right-hand ComboBox, as below.



Commonly used properties of the DateTimePicker control

CalendarFont The font in which the rolled-down calendar will be displayed at run time. This property does not change the font in the ComboBox aspect of the control.

CalendarForeColor
The colour of the date numbers on the rolled-down calendar at run time.

CalendarMonthBackground
The background colour of the rolled-down calendar at run time.

CalendarTitleBackColor
The background colour of the banner containing the month name in the rolled-down calendar at run time.

CalendarTitleForeColor
The colour of the text containing the month name in the rolled-down calendar at run time.

CalendarTrailingForeColor
The colour of the dates of the preceding and following months that appear in the current month's rolled down calendar at run time.

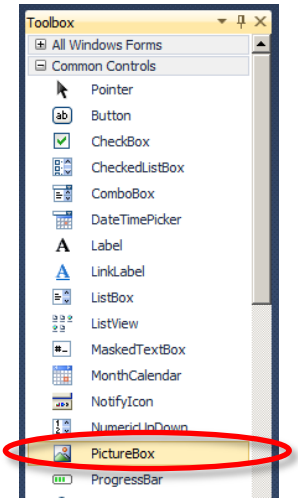
MaxDate The maximum date that will be displayed in the DateTimePicker at run time. If this date is within the current month then any dates after that will not be displayed in the rolled-down calendar.

MinDate The minimum date that will be displayed in the DateTimePicker at run time. If this date is within the current month then any dates before that will not be displayed in the rolled-down calendar.

Value The selected date in the DateTimePicker control at run time. The format of this value field is "m(m)/dd/yyyy hh:mm:ss". The default for the time aspect of this value is 00:00:00. If the month number is a single-digit figure (January through to September, for example) then there is no leading zero.

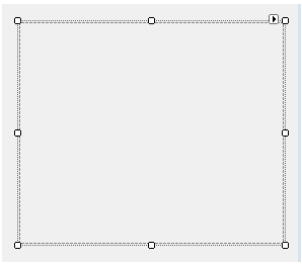
PictureBox

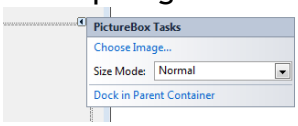
The PictureBox control is accessed from the Common Controls section of the Toolbox by clicking on the icon shown here.



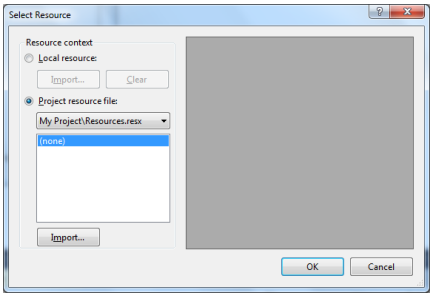
The way to get a PictureBox onto a form is to click on the PictureBox icon in the Toolbox (shown here), and then move across to the form with it. When you reach the place where you want the top left-hand corner of the PictureBox to be, press the left mouse button and drag across and down to describe the size of PictureBox that you need. Release the left mouse button when you're happy.

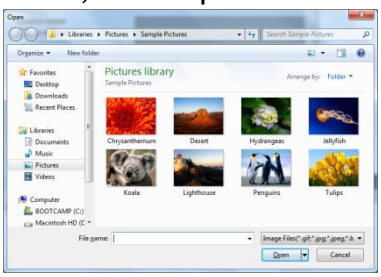
Once a PictureBox control has been added to a form it is possible to resize it by using the grab handles around the edges, which are delimited by dotted lines. Notice on the illustration to the right that towards the right corner of the top edge of the PictureBox that there is a small square with an arrow within. By clicking on this arrow a side menu opens from which it is possible to select the image the PictureBox is to contain, the size mode of the image and whether you wish the image to dock within the parent control. In the example here the parent control (i.e. the control in which the PictureBox is placed) is the form.

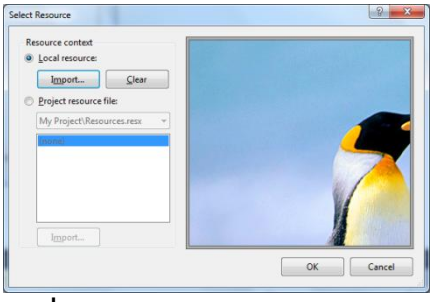




To insert an image into the PictureBox, either use the side menu shown above, or click on the ellipses to the right of the Image property in the Properties Window. The "Select Resource" dialog box opens, as shown here to the right. Click on the "Local resource" radio button, then click the "Import" button. From there, the "Open" screen opens, allowing you to navigate through your computer files to locate the picture of your choice. Once you have found your picture, click "Open". In the illustration to the right notice that I have selected to add the picture called "Penguins" to my PictureBox. Click the OK button.







However, the image is not fully included due to the fact that it is too large to fit within the size of PictureBox that we have to work with. To allow the entire image to display without resizing the PictureBox, click on either the small arrow to the top right edge of the PictureBox, or go to the SizeMode property in the Properties Window – both give you to the same options. The default is "Normal" which keeps the PictureBox and image to their original size – thus losing every part of my penguin except its right ear. "StretchImage" allows the entire image to be displayed – however, if the PictureBox is not the correct size then the image will be distorted and will need to be manually altered.

Commonly used controls – PictureBox control

Lecturer: Jane Fletcher

SizeMode of Normal at run time

SizeMode of StretchImage at run time

SizeMode of AutoSize, CentreImage and Zoom at design time – note the lines describing the size that tie PictureBox would need to be to accommodate the image

The default event of the PictureBox object is the Click event, although there are many others available that you could explore. It is unusual to assign an event to a PictureBox, however – generally they are used as a display other than as a dynamic object (one that does something or causes an event to happen).

The Dock property of the PictureBox object allows the designer to decide where the PictureBox should be placed within the form. The default value for this property is "None". There are other options available – "Top", "Left", "Fill", "Right" and "Bottom". See below to see the effects these different values have on the PictureBox at run time.

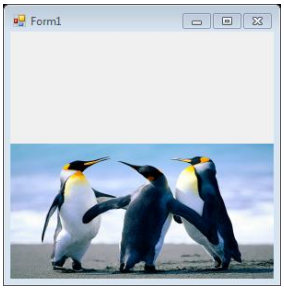
1

2

3

4

5



6

	Dock Property
1	None
2	Top
3	Left
4	Fill
5	Right
6	Bottom

Commonly used Properties of the PictureBox control

BorderStyle What the borders of the PictureBox will look like at run time. Options are None (the default), FixedSingle (a line drawn around the edge of the PictureBox) and Fixed3D (an embossing effect around the edges of the picture).

Dock Where the image will be placed in its 'host object' if this property is set to anything other than 'None' (see above).

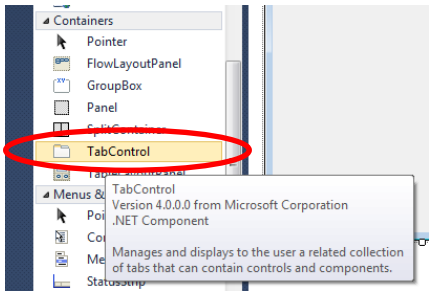
Enabled Whether or not the PictureBox is active – if there are any functions based around the PictureBox (like a Click event) and this property is set to False, then the event / function will not be accessible. The reverse is true if this property is set to True.

Image The location of the image to be uploaded at run time to this PictureBox object.

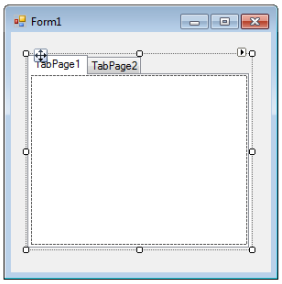
SizeMode Options available are Normal, StretchImage, AutoSize, CentreImage and Zoom. If you want the image you're uploading to fit to size then you should select StretchImage. There may be some aspect ratio distortion if you do this, however.

TabControl

The TabControl object gives you an array of 'pages' upon which you can place other objects. When the corresponding tab is clicked by the user, that page comes to the top and becomes active. Each of the different 'tabs' act like a separate part of the form, despite the fact that they sit on top of each other.



The way to get a TabControl object onto your form is to select it from the 'Containers' section of the Toolbox, shown here, and then move across to the form with it. When you reach the place where you want the top left-hand corner of the TabControl to be, press the left mouse button and drag across and down to describe the size of TabControl that you need. Release the left mouse button when you're happy. It is possible to resize the TabControl by using the grab handles that are located around the borders of the object.



In the image to the right a TabControl object has been added to a form. Notice that there are, by default, two TabPages within the TabControl. Their default names are TabPage1 and TabPage2.

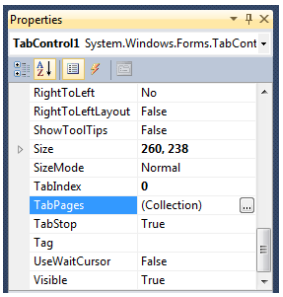
TabControls require 'drawing' using the DrawItem method of the object, as shown in the sample code below. This TabControl has been called tbMenu in its Name property, and the programmer has selected the DrawItem method, as shown in the 'Private Sub' routine title.

```
Private Sub tbMenu_DrawItem(ByVal sender As Object, ByVal e As _
            System.Windows.Forms.DrawItemEventArgs) Handles tbMenu.DrawItem

    'This subroutine is called every time the tabs are accessed.
    'Firstly we'll define some parameters. These are used to "draw" the tabs.
    'Tabs aren't objects like we know them - they are "drawn" fresh each time we use them.
    Dim CurrentTab As TabPage = tbMenu.TabPages(e.Index)
    Dim ItemRect As Rectangle = tbMenu.GetTabRect(e.Index)
    Dim FillBrush As New SolidBrush(Color.Black)
    Dim TextBrush As New SolidBrush(Color.White)
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    'If we are currently painting the Selected TabItem we'll
    'change the brush colors and inflate the rectangle.
    If CBool(e.State And DrawItemState.Selected) Then
        FillBrush.Color = Color.White
        TextBrush.Color = Color.Black
        ItemRect.Inflate(2, 2)
    End If

    'Next we'll paint the TabItem with our Fill Brush
    e.Graphics.FillRectangle(FillBrush, ItemRect)

    'Now draw the text.
    e.Graphics.DrawString(CurrentTab.Text, e.Font, TextBrush, _
          RectangleF.op_Implicit(ItemRect), sf)
```

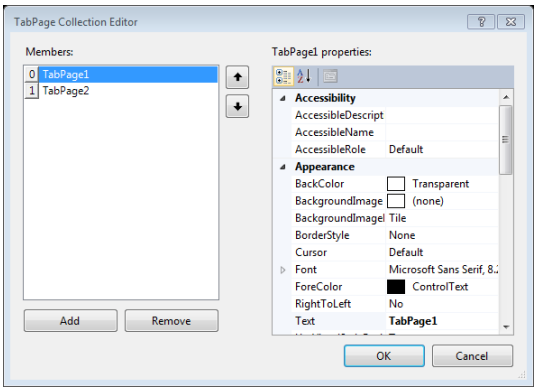
```
        'Reset any Graphics rotation
        e.Graphics.ResetTransform()

        'Finally, we should Dispose of our brushes.
        FillBrush.Dispose()
        TextBrush.Dispose()
    End Sub
```

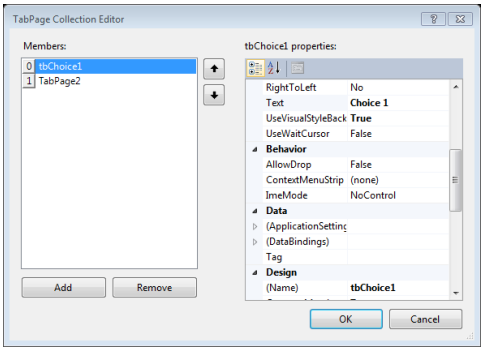
This will all be explained more fully later on in this document when you will actually write a program that uses the TabControl object (see Exercises 10.0 and 10.1).

The way that different pages are added to a TabControl is via its TabPages property. This can either be done at design time or run time. In the example below we are going to add some tab pages at design time.



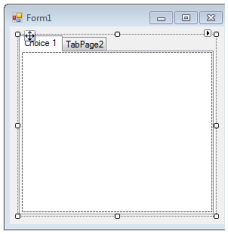
Within the Properties Window for the TabControl, click on the TabPages property and then on the ellipses that will appear to the right of the 'collection'. Click on the ellipses. The TabPage Collection Editor will appear, as shown in the illustration below. Notice that the properties are displayed in category order – you can alter this to alphabetical order if you prefer it, by clicking on the "A-Z" button. To the left of this form you have a panel that lists the two TabPages that are currently within this TabControl. We are going to alter them to tbChoice1 and tbChoice2 with Text properties of "Choice 1" and "Choice 2". In the 'Appearance' part of the properties, change TabPage1's Text property to "Choice 1". Then within the 'Design' section of the Properties window change the Name property to tbChoice1. The result of taking this course of action is shown in the illustration to the right. Notice that on the left-hand side the first page (0) is now called tbChoice1 and the Text property will display "Choice 1".

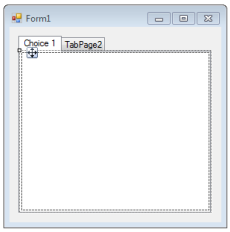




One thing to be aware of when using TabControls is the fact that you have to make sure that you're selecting the right part of the object when you're double-clicking on it to get to the Source Code window. Notice the difference between the two images below; the first image shows the entire TabControl as being active, whereas the second one shows the first TabPage (tbChoice1) as being active. You will get very different results by clicking on each of these!

TabControl active





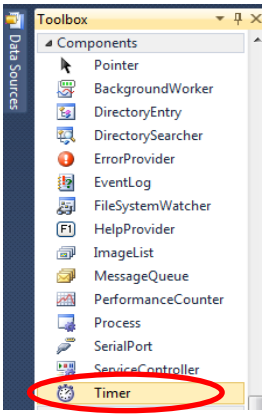
TabPage active

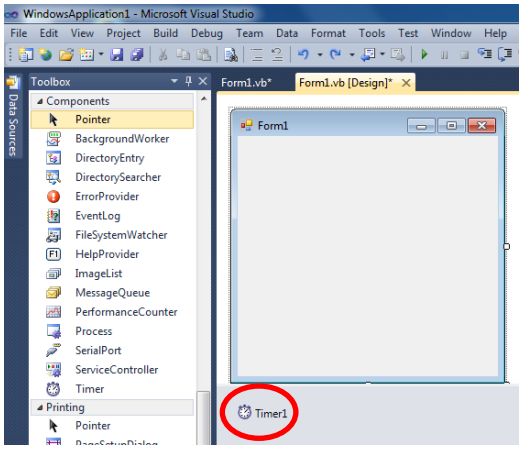
Commonly used Properties of the TabControl object

DrawMode	Allows the developer to change the colours of the tab, if this property is set to "OwnerDrawFixed". If this property is not changed from its default value of Normal then the appearance of the tab cannot be changed.
Font	The font size and style that will appear in the tab.
TabPages	This is a Collection of pages that will fit within the one TabControl. See above for a longer description on the use of this property.

Timer

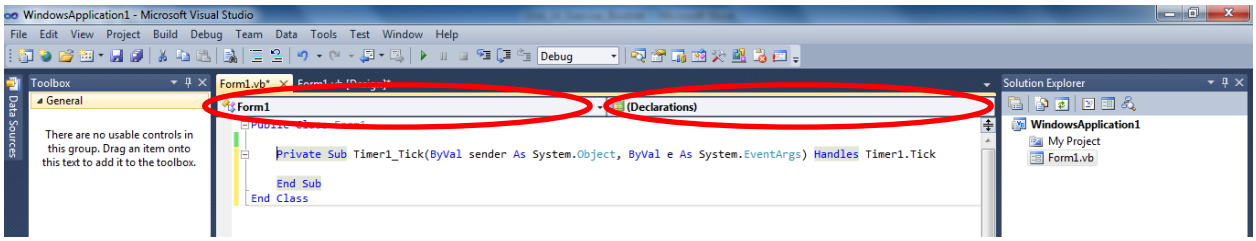
The Timer control is accessed from the Components section of the Toolbox by clicking on the icon shown here. It is the last one of the Components section. The way to get a Timer control onto your form is to left click on the icon in the Toolbox, then move across to the form and left click once on the Form object. The Timer control does not appear on the form itself - wherever you choose to place it on the form it will always move and will appear in the grey area below the Form object, as shown in the image on the left. Do not make the mistake of adding lots of Timers in the hope of getting one to attach itself to the form!





In order to gain access to the default event of the Timer object, double-click on the Timer itself. The default event of the Timer object is the Tick event, but the default timing between the Ticks is not set to 1 second - the default timing is 1/10th of a second, or an Interval of 100. An Interval of 1000 is equal to 1 second in real time. You can change the Interval in the Properties Window of the Timer control.

There are many more Events that can be used with the Timer control, and can be accessed via the pull-down list of events shown in the red highlight, below.



The way that an enabled Timer_Tick event works is that it is triggered each time the Interval is reached. If the Interval property of the Timer is set to 1000 then the Tick event will occur every second and whatever code occurs within that Tick event will be executed every second. Bear in mind that this could easily cause endless loops and tie up your computer's processing memory! It is important to remember that unless you want your Tick event to run constantly (if, for example, you have a clock running on your form that you want to update every second) that you disable the timer at the appropriate point. It is possible to alter the Enabled property at run time or design time, or a combination of the two.

A Timer's Tick event can be active throughout the running of a program; the Timer itself doesn't have to be accessed at all to cause it to trigger the Tick event unless it is to either start the Tick event procedure or to stop it. It is happy to run in the background without intervention but it can make debugging a program step-by-step very difficult.

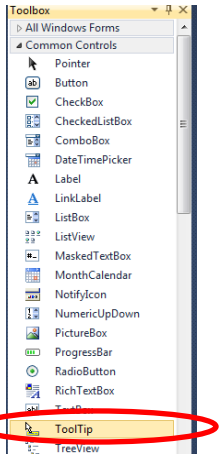
Commonly used Properties of the TabControl object

Name The prefix for the Timer object is tmr.

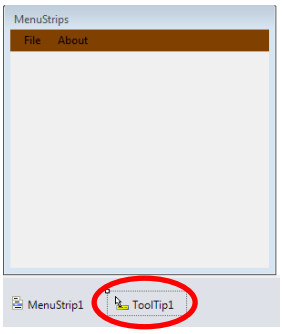
Enabled The default value for the Enabled property is False. Set this property to True when you want the Timer object to be active.

Interval The default value for the Interval property is 100, which equates to 1/10th of a second. One second requires the Interval property to be set to 1000; five seconds requires an Interval setting of 5000 and so on.

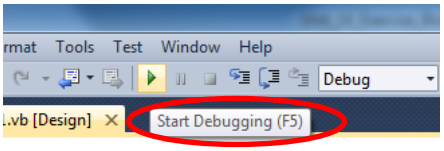
ToolTip



The ToolTip control is accessible via the Common Controls area of the Toolbox. The ToolTip doesn't have to be placed on a form but can be defined within the code; either method works just as well. The author's preferred method is to declare within the code. If you choose to click on the Tooltip item in the Toolbox's Common Control menu then place it on the form but it will go to the grey pane beneath the Form object's template.



The purpose of the ToolTip object is to display context-sensitive information to the user at run time. Microsoft Visual Studio has useful ToolTips, as shown in the screenshot below.



The ToolTip object here is activated when the user (in this case, the developer) hovers over the debugging icon in the icon bar. How long it takes to get onto the screen once the mouse is in position, and how long it remains on the screen whilst the user remains in position, is determined by the developer within code.

The first thing to do when establishing a ToolTip is to declare it. This is not essential if you have placed a ToolTip object on the form in design mode.

```
'Set up the tooltips
Dim toolTip1 As New ToolTip
```

The coding above declares the ToolTip object. It is done within the Form_Load event, which you reach by double-clicking on the Form object from the Form editor.

The coding below also occurs within the Form_Load procedure, and sets up timings that apply to the ToolTip object. There are three aspects to the timing, reading in order down the code below: how long you have it on screen (AutoPopDelay), the delay before the ToolTip text appears (InitialDelay) and the delay before you reshow the ToolTip (ReshowDelay). Timings are based on 1000 being equal to one second.

```
' Set up the delays for the ToolTip.
toolTip1.AutoPopDelay = 5000
toolTip1.InitialDelay = 1000
toolTip1.ReshowDelay = 500
```

The ToolTip will not appear automatically, so that needs to be coded too, in the Form_Load event:

```
' Force the ToolTip text to be displayed whether or not the form is active.
toolTip1.ShowAlways = True
```

Assign the ToolTip to work for the MenuStrip as well (also in the Form_Load event):

```
MenuStrip1.ShowItemToolTips = True
```

Once the setup has been done by following the statements above, the ToolTip is ready to be assigned to each of the objects on the form that you think requires contextual help. You do not have to have a ToolTip for each object – just the one will do.

The format of the statement to assign contextual text relate the object to the ToolTip goes as follows:

***NameOfToolTip*.SetToolTip(Me.*ObjectName*, "Text to display for *ObjectName*")**

where you replace the *NameOfToolTip* with the .. erm .. name of the ToolTip, and *ObjectName* is the name of the object that requires the contextual help. Ensure that you use explicit help, and be polite at all times.

In the example below, three buttons representing three different users, are assigned to the toolTip1 object, again within the Form_Load object.

```
        toolTip1.SetToolTip(Me.btnUser1, _
"Click here if you have been assigned User Code 1. If you are unsure, check with your manager.")
        toolTip1.SetToolTip(Me.btnUser2, _
"Click here if you have been assigned User Code 2. If you are unsure, check with your manager.")
        toolTip1.SetToolTip(Me.btnUser3, _
"Click here if you have been assigned User Code 3. If you are unsure, check with your manager.")
```

Assigning ToolTip text is time-consuming but is extremely useful for assigning onscreen help for the user (see the requirements of the P3 criterion of Unit 14 – Event Driven Programming).

Exercise 1

1. What do the initials “IDE” stand for?
2. Within a visual programming environment, what does the term “object” mean?
3. Name four different objects that can be put onto a form to help build a graphical user interface.
4. What is the Solution Explorer used for?
5. What object would you use on a form if its only purpose was to display information to the user?
6. What is the default event for a Button object?
7. I’ve just typed “if you want to enter this program what you need to do is to click on the red button that says enter” into the Text property of my label. All I can see on it is “if you want to enter th” before it all disappears at the edge of the form. I’ve tried to make my label bigger but it won’t grow! What have I done wrong? (Tip: the answer isn’t “started programming in the first place!”)
8. How do I make a different style of typeface appear on every object on a form?
9. Name four properties that will change the appearance of a label.
10. How do I stop the graphical user interface from appearing anywhere it likes on the computer screen when I run the program?
11. How do I start to internally document a program?

Exercise 2

1. What property do you change on a button to stop the user being able to click on it, but with it still being visible?

2. What object could a programmer use to cluster together a group of related objects?

3. When is it allowable not to rename an object from its default name?

4. Where would you find the executable version of your program, if you were using Windows explorer?

5. Name three events that you can use with a Button object.

6. My text box won't allow me to stretch it downwards to enter more text. What property do I have to change?

7. What two ways can I use to make sure that the cursor is in the text box I want it to be in when the program starts running?

8. Give reasons why you must always internally document a program.

9. We haven't discussed this in class yet, but think about things that the user could do wrong when asked to enter information into a text box, and list as many problems that you can think of that a programmer could consider when writing reliable and non-crashworthy code.

10. You have to write a program that has four buttons side by side, each performing a similar task. You have to make sure that the user can't use these buttons before certain criteria have been fulfilled. Without going through and changing the properties of each button in turn in the code, what can you do to make just one line of code stop each of the four buttons from working?

Exercise 3

1. What is the prefix for a radio button?
2. How do you ensure that a group of radio buttons work together as a team?
3. What is the default event for a radio button?
4. What is the essential logical difference between a radio button and a check box?
5. What type of variable do you need to declare in a program if it is to hold whole numbers only?
6. Declare one if the variables mentioned in question 5 – any name as long as it follows all the rules.
7. What is the purpose of a program specification?
8. Why does a program need design documentation such as form designs and form dictionaries?
9. What is the difference between technical documentation (such as program specifications) and user documentation?
10. Give three different ways that a numeric value should be validated.

Exercise 4

1. What is the prefix for a list box?
2. What is the difference between a list box and a combo box?
3. What Visual Basic .NET function would you use to determine whether a particular input contains numbers or not?
4. What must you ensure before you execute a "lstbox.Items.RemoveAt" command?
5. Give two situations where you would use a group box.
6. What is the default event for a list box?
7. When would that default event be activated?
8. What must the last words of any program be?
9. What is the purpose of a form dictionary?
10. What properties must always be named in a form dictionary?

Exercise 5

1. What is the prefix for a check box?
2. What is the default event for a check box?
3. What values can be contained in a variable of the type “string”?
4. What validation can be used on a text box that should contain string input?
5. Is it possible to put a combination of check boxes and radio buttons into one group box and still have the radio buttons work together properly?
6. Give three different ways of getting user input.
7. Write the concatenation statement to put together the following: the contents of the text box txtForename together with the text box txtSurname, followed by the literal “lives in Nottingham.” Into the variable strStudentInformation.
8. What is the meaning of the word “literal” in a programming context?
9. For which programs written in this unit should you create design documentation such as form dictionaries?
10. What is the most difficult program you have written so far? Why?

Exercise 6

1. Name three objects that could be used to gain input from the user.
2. Name three objects that could be used to give output to the user.
3. Is a label a good object to use to get input from the user? Explain your answer.
4. Name three ways that a number be validated when it is input to a program.
5. Explain when you would use a radio button rather than a check box and give an example when a radio button is better. Give an example where a check box would be better.
6. Name the three control structures. (Think "Wednesday's lesson"!)
7. Name three types of variable that can be used in any program.
8. Explain why you would use each of them rather than any other type of variable.

Creating a folder structure

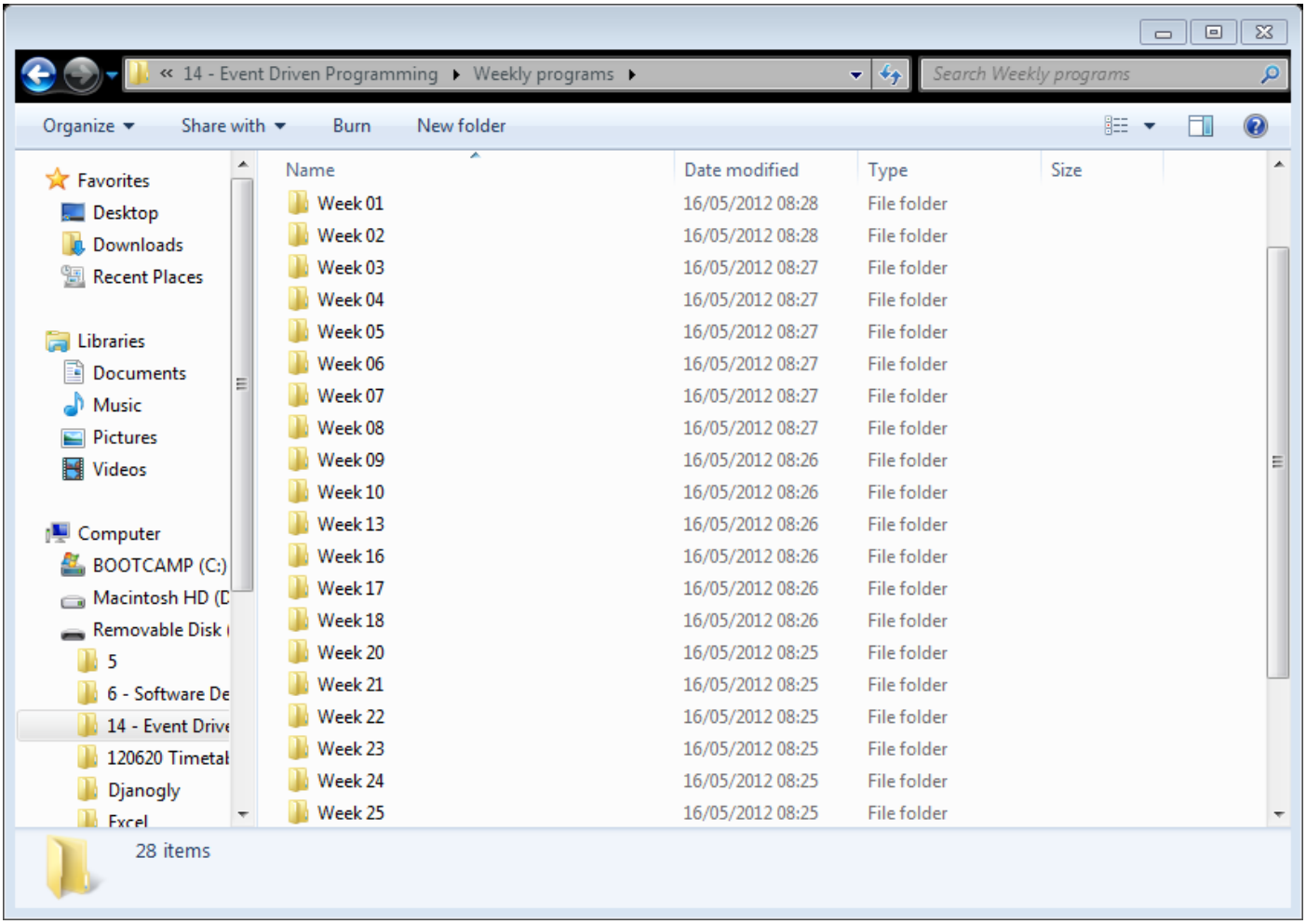
Lecturer: Jane Fletcher

This section will explain how to set up a folder structure to enable you to locate your projects easily.

Every project that you create MUST have its own folder. If you are going to use a previously written project as the basis to develop a further project that has different features, then you should copy the entire folder of the original project and paste it with a new name.

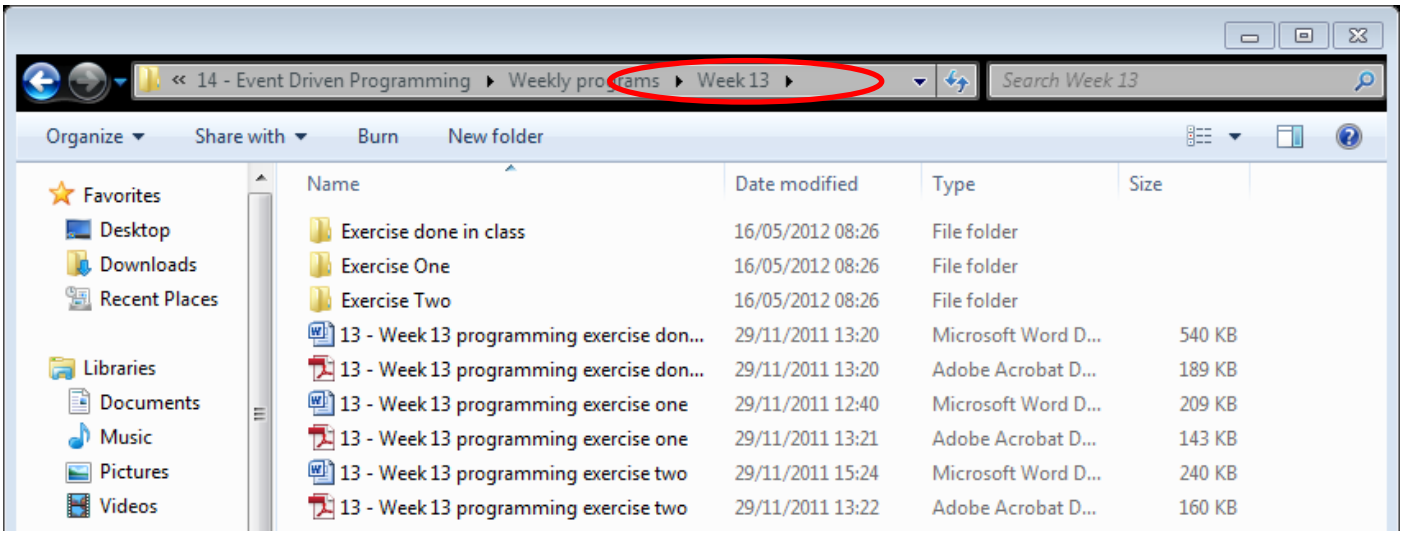
Before you start thinking about creating a new project, you must first design a file system where you will be able to readily find each project you have made, within an easily identified folder specifically for this unit. There are various ways of doing this: the first (and easiest) way is to create a different folder for each week of the course, and within each week's folder have separate folders for each different project. Another way is to name your folders with a readily identified name that states the purpose of the project contained within it.

Below is a screenshot of the file structure adopted by the author of this document. Note that the "master" folder is called "14 – Event Driven Programming", then there is a further folder called "Weekly programs", and thereafter there are folders for each week of the course.

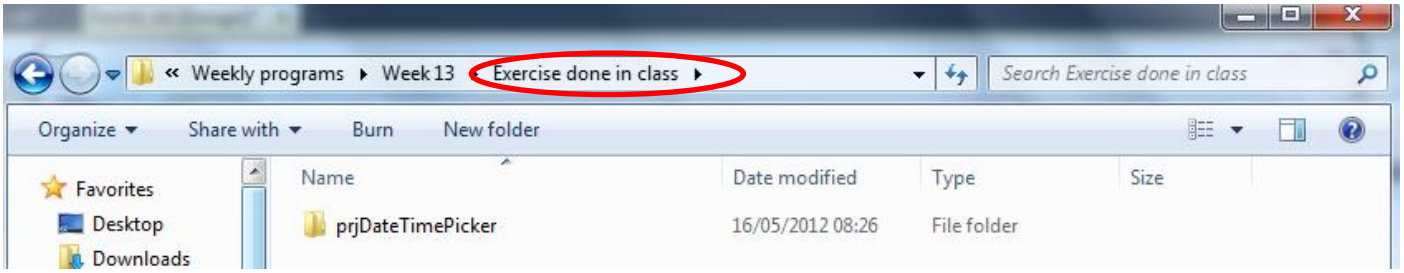


Creating a folder structure

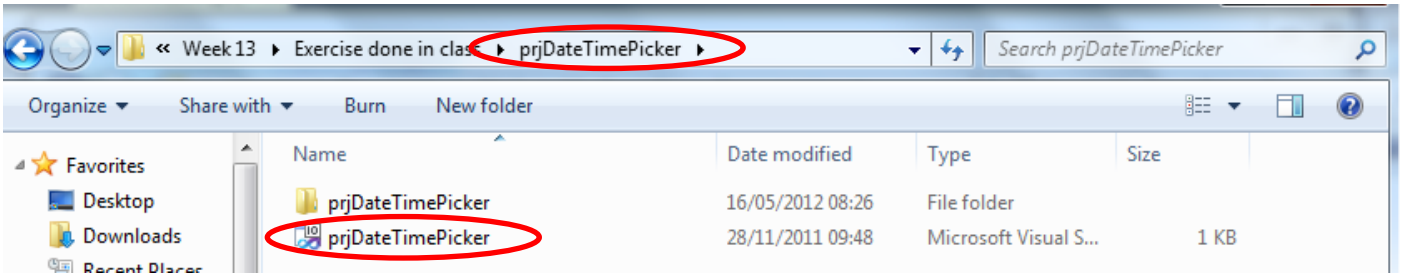
Next, here is a screenshot of week 13 of the course:



Notice the top three folders contained in the subfolder: “Exercise done in class”, “Exercise One” and “Exercise Two”. These folders were created by the author to separate the three different programs to be written during week 13. Next we will open the folder “Exercise done in class”, shown below.



Folder “Exercise done in class” contains a sub-folder called ‘prjDateTimePicker”, created by the Visual Basic .NET environment. This was not created by the author.

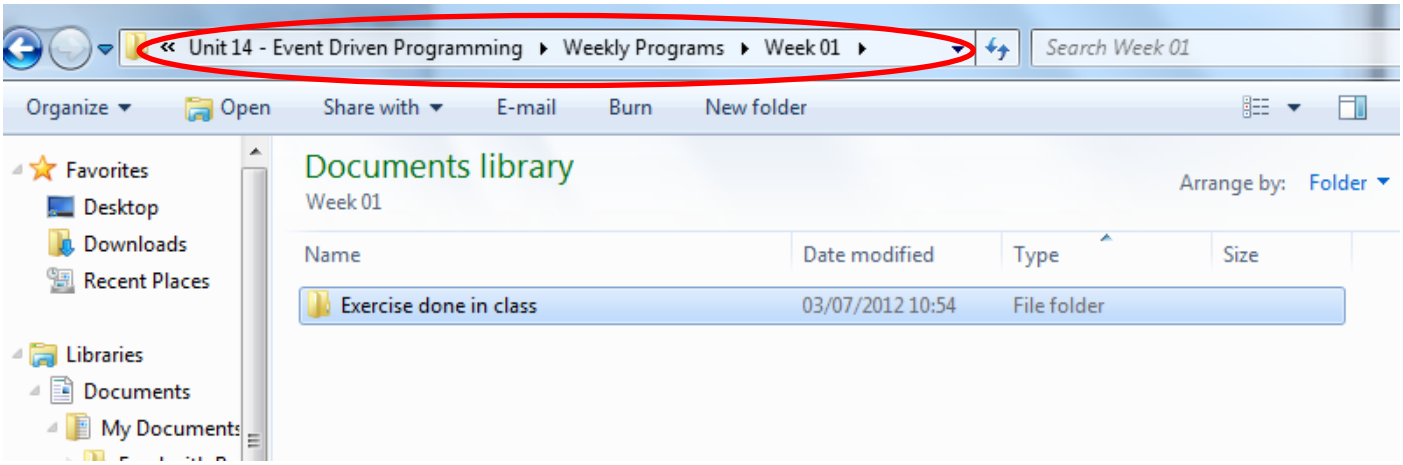


This folder contains a further folder, and a Microsoft Visual Studio Solution file called “prjDateTmePicker”. It is always this Solution file that should be opened when a previously created project is opened – this is the ‘front door’ to the project and gives the .NET framework access to all the different files created as part of the original and edited project. If you are accessing a previously written project via Windows Explorer, NEVER open the project by clicking

on any other file within the project because doing so stands an excellent chance of corrupting your work!

Let us assume that you are on Week One of the course and need to start creating your file structure.

1. Within your "My Documents" folder, create a new folder entitled "Unit 14 – Event Driven Programming".
2. Within this folder, create a folder called "Weekly Programs".
3. Within this folder, create a folder called "Week 01". Note the zero before the 1, which will ensure that week two's folder comes before week 11's folder.
4. Now create a further folder, and call it "Exercise done in class". This is the folder where we will be putting our first program.



Your folder structure should look similar to the one shown in the illustration above.

You are now ready to start developing your first project!

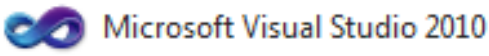
Starting a new project

Lecturer: Jane Fletcher

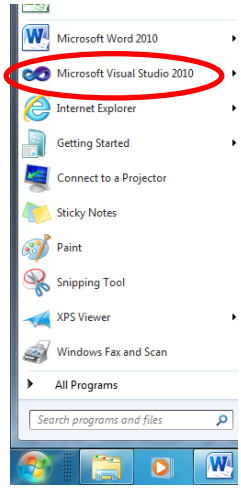
The first thing you are going to have to do is to load the Visual Studio .NET software. Depending on the setup of your computer, you will find it on the following path:

Start → All Programs → Microsoft Visual Studio 2010 → Microsoft Visual Studio 2010

The icon for the Visual Studio software looks like the illustration to the right and this is the one that you must click in order to get Visual Studio to load.

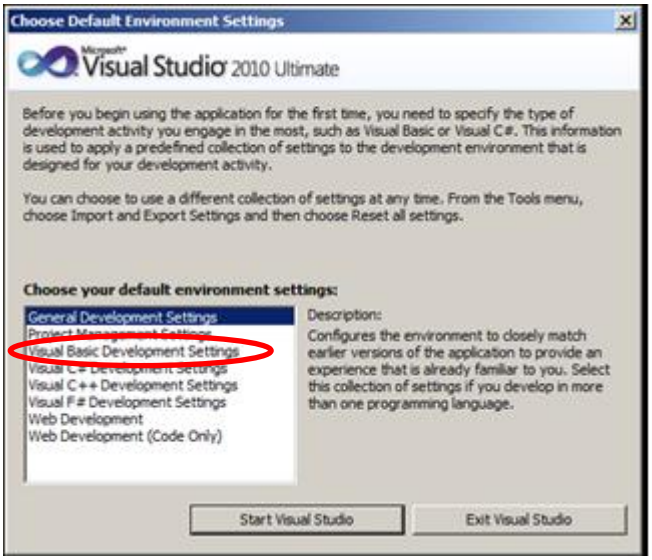


You may find that on subsequent loadings of the software it will appear in your list of program "favourites" when you click your Start button, as shown here to the right.



Once you have clicked Microsoft Visual Studio 2010, the software's splash screen will load, shown below.

When the software has loaded, the first time you run Visual Studio you may be shown a screen that asks you to choose your default environment settings, as shown below.



Make sure that you select "Visual Basic Development Settings". This is the one circled in red, rather than the one that is selected in the illustration shown to the left. Once you've done that, click the "Start Visual Studio" button.

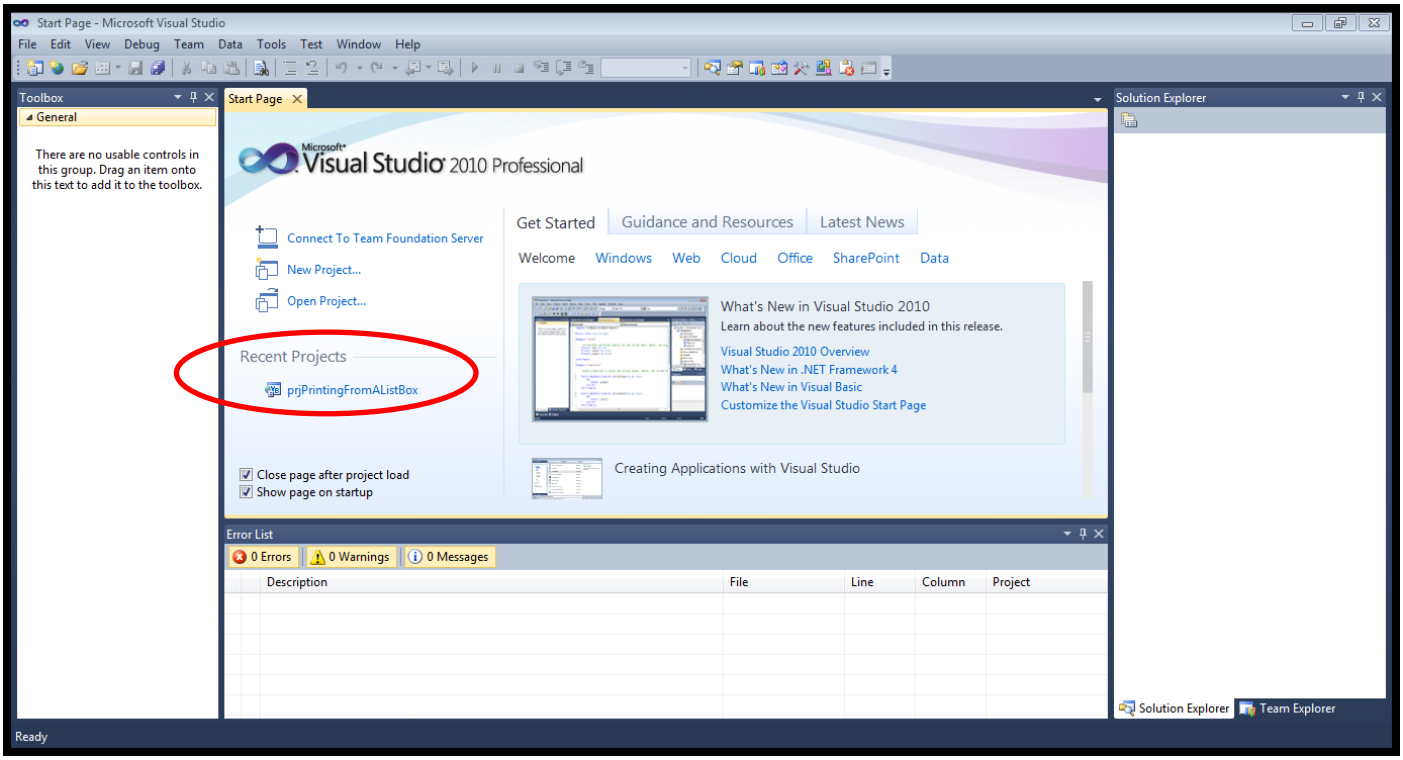
You will probably not be asked to do this again on a College computer, but if you are make sure you always pick Visual Basic Development Settings. It isn't the end of the world if you make a mistake, so don't panic if that is the case. It will mean, however, that you will be presented with an entirely different development environment which could cause panic!

Once you have clicked on the "Start Visual Studio" button, you are entering the world of VB .NET!

Starting a new project

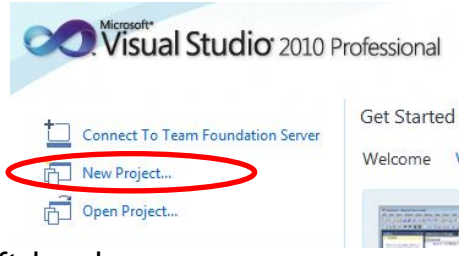
Lecturer: Jane Fletcher

The next thing you will see is the Visual Studio 2010 Professional Start Page for the Visual Basic .NET environment. This screen is shown below. Yours will not look exactly like this, but will be very similar. The section of the screen shown in the red circle below will be empty the first time you run Visual Studio because it is a list of your recently accessed projects. You can see from the illustration below that the last project accessed on this computer by the author is one entitled 'prjPrintingFromAListBox'. When you have written a few projects you will find them listed in this area of the form. It isn't always wise, however, to rely on this aspect of the software to enable you to find what you're looking for – you're far better off having a robust file structure!

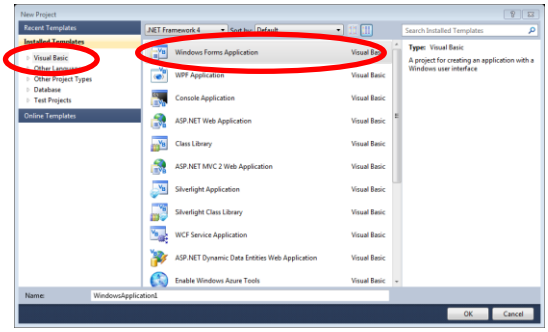


Before you can begin to write your first program you have to load the correct editor that you will need in order to create a project that contains Windows forms.

Click on "New Project".



You will be invited to select from recently accessed templates, as shown in the illustration below.

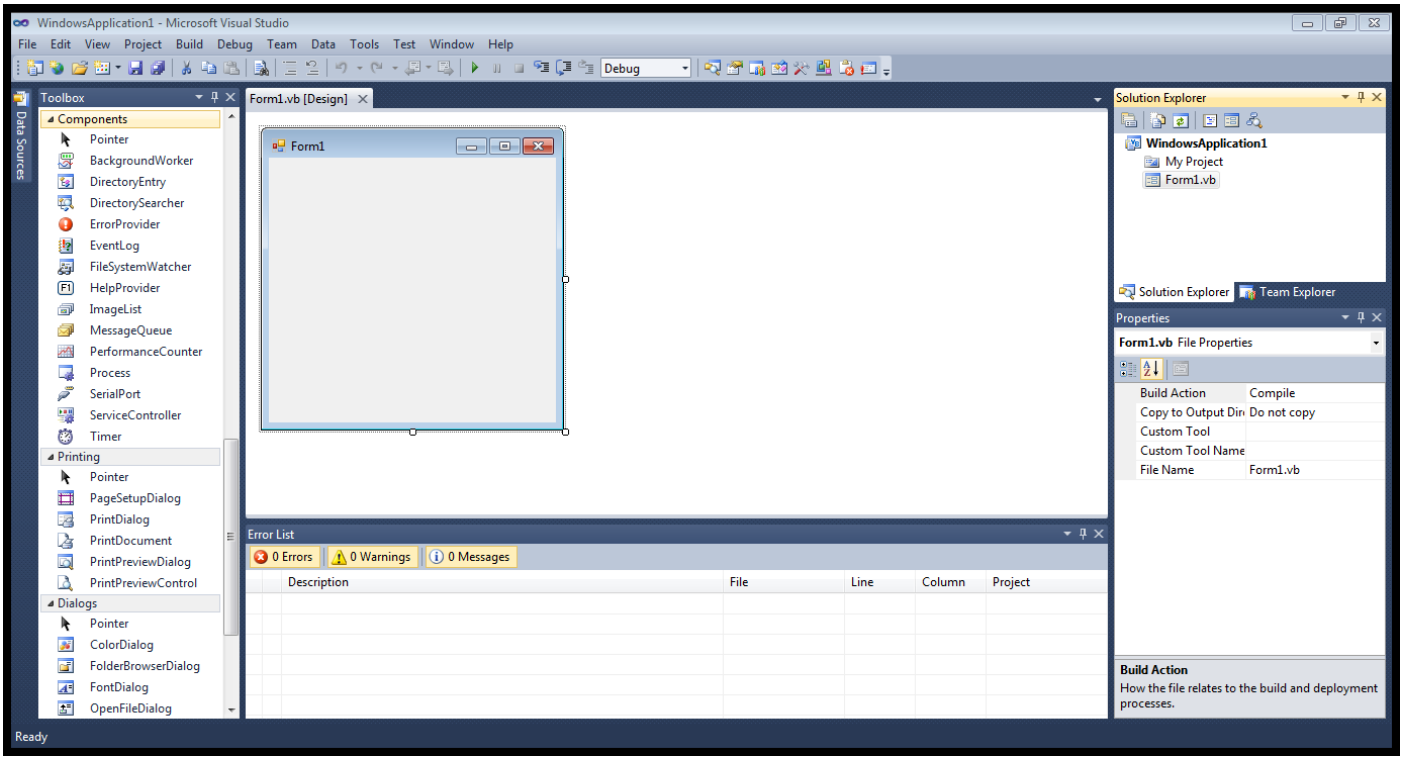


Make sure that "Visual Basic" is selected in the left-hand pane.

The central area of the form shows the different types of template that you can access. Most of the projects you will create require you to access the top one, "Windows Form Application". Select that.

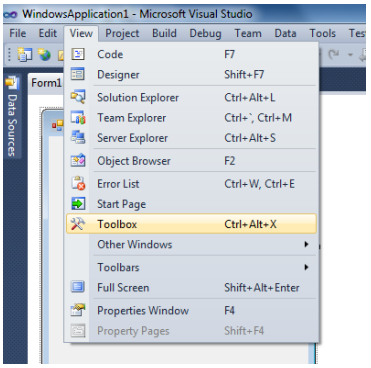
Starting a new project

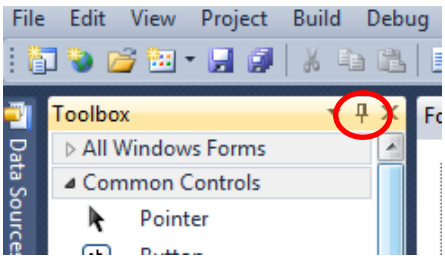
Lecturer: Jane Fletcher



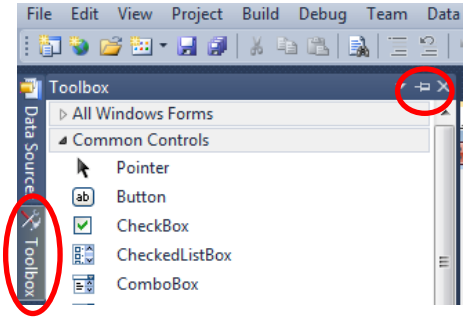
Above, you can see the Form Editor or Design View of VB .NET. If you turn to page 6 of this document you will see a breakdown of what each aspect of this screen does. To the left you will see the Toolbox.

If you do NOT see the Toolbox, click on "View" in the menu bar, and then select "Toolbox". This is shown to the right. You can also click Ctrl+Alt+X together and it will do the same thing – give you your toolbox back.





Notice on the Toolbox that there is a picture of a drawing pin on the top line – in the illustration below it is pointing downwards. This drawing pin 'holds' the Toolbox in place. If you click on it, the Toolbox will hide on the left-hand side of the screen and will only be visible when you hover over the 'Toolbox' tab to the left of the screen. The drawing pin lies on its side during this view. To get the Toolbox back 'permanently' again, click on the recumbent drawing pin and it will become upright and hold the Toolbox in place.



Before you do any work on your project, you must save it in the location of your choice (i.e. in the folder system you made back on pages 51 to 53).

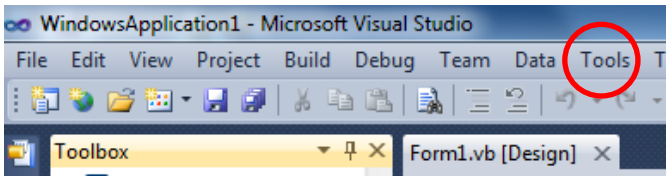
Saving a project

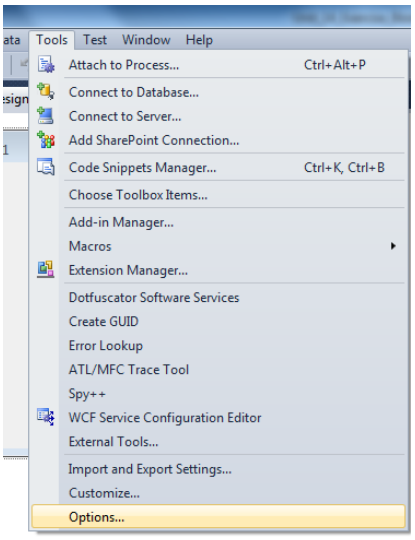
Lecturer: Jane Fletcher

Visual Studio has been set up on the College system to default to saving your projects in the "Visual Studio 2010" folder in "My Documents". This can be very annoying if you've already established your own folder system (as explained in this document) and want to save your projects somewhere else completely.

The following section will show you how to change your default settings for where your projects will be saved. You do not HAVE to do this, but it saves time and effort later on if you choose to do it.

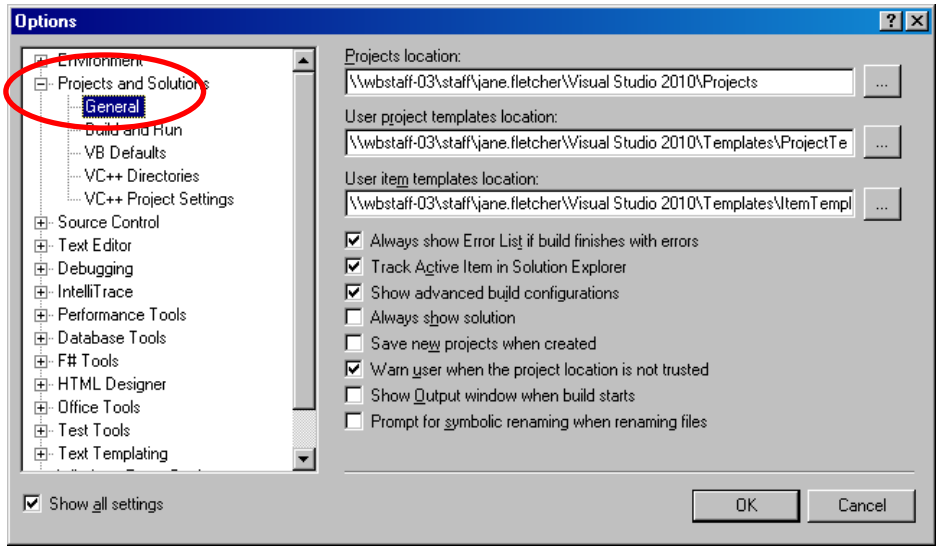
Click on "Tools" in the menu bar.





From the Tools menu, select "Options".

Make sure that "Projects and Solutions" is selected in the left-hand pane of the subsequent dialog box, and within that, "General".

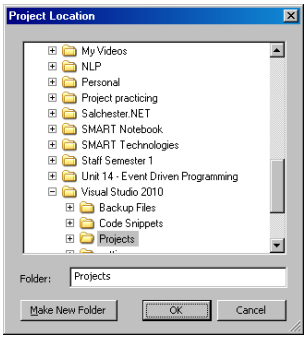


Notice that to the right of the Options list are three combo boxes, each with ellipses. The top one of these three is headed "Projects location". Detailed there is the long and complex path of where projects will be saved for this author on the College network, by default. It says:

\\wbstaff-03\staff\jane.fletcher\Visual Studio 2010\Projects

This is to be changed to enable saving to take place in the "Unit 14 – Event Driven Programming" and "Weekly Programs" folders.

Click on the ellipses to the right of the named project location. The Project Location dialog box will open – shown on the right.



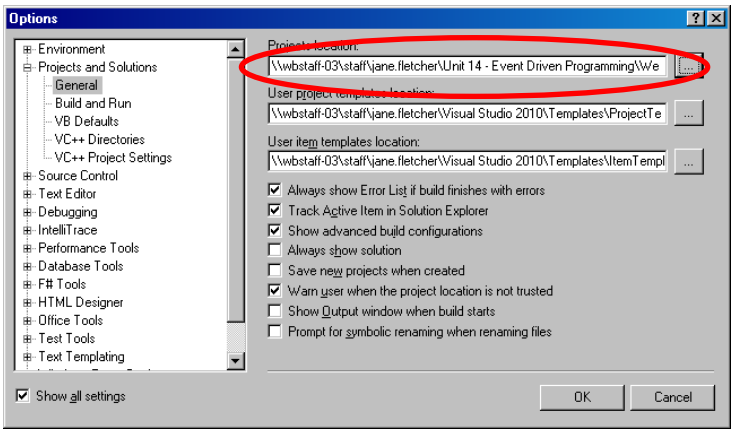
Notice that the default saving location has been set at a path of

My Documents → Visual Studio 2010 → Projects

Saving a project

Lecturer: Jane Fletcher

Click on the folder that you made earlier, "Unit 14 – Event Driven Programming", then "Weekly Programs", then click on the "OK" button. Once you have done this, the Options dialog box will show you that your new location has been selected.

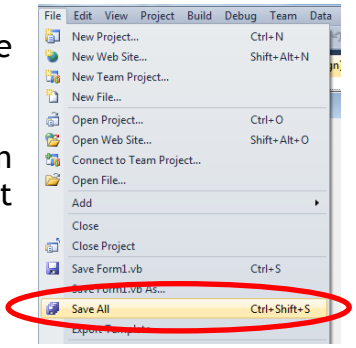


Click the "OK" button to continue.

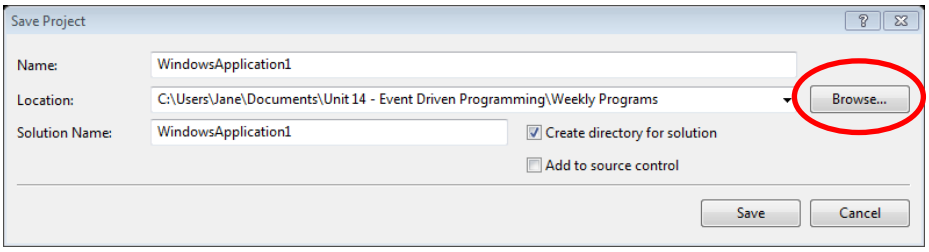
You will still have to make sure that you select the correct week's folder when you save for the first time, which will be explained below.

You should now be back at the Visual Basic .NET design view. You haven't yet saved your project; you have just set the default location path for when you actually DO save.

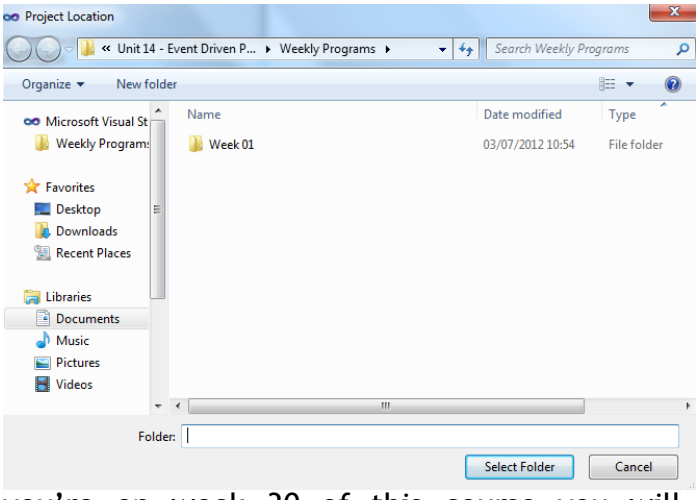
To save your project, click on "File" from the menu bar and then "Save All".



The "Save Project" dialog box will now open. Notice that the location has changed to the path that you selected above, but is still only at 'Weekly programs' and not 'Week 01'.

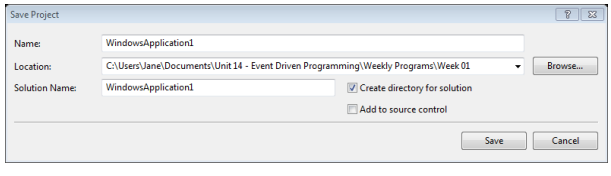


Click on the "Browse" button to go further down the "Weekly Programs" path.



The "Project Location" dialog box appears. From here you can click on "Week 01" and then click "Select Folder". Notice that "Week 01" has been added to the Location text box. If you have 'Exercise done in class' as a folder, open that one too.

The project name is shown in the first text box – the default name is "WindowsApplication1". This is NOT a good project name because when you're on week 30 of this course you will have upwards of 60 of these so you need a way to distinguish one from another.



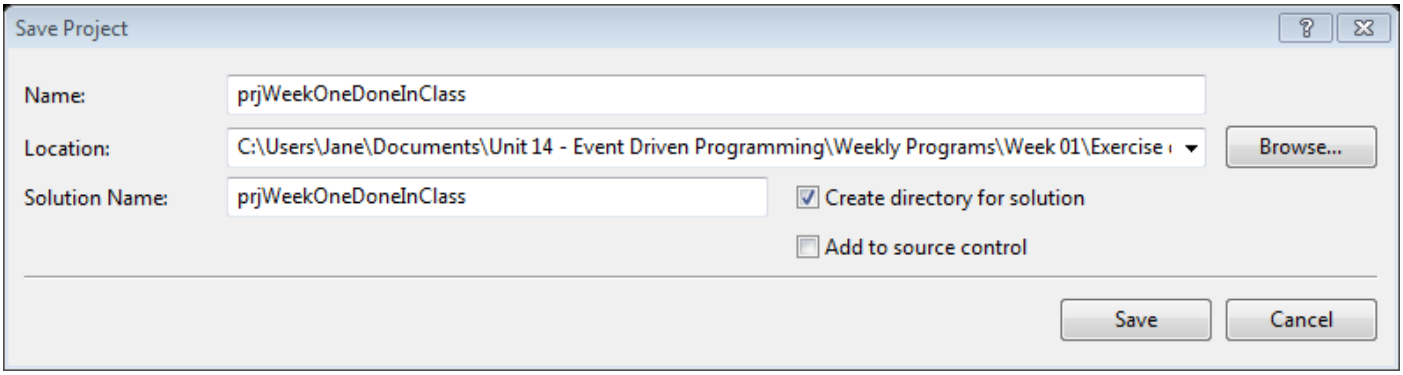
Saving a project

Lecturer: Jane Fletcher

A project's prefix is "prj". You should always use the i-capping rule for naming your project.

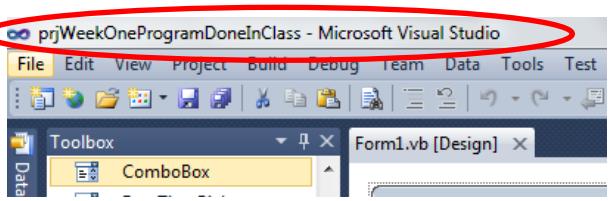
In this example, we are going to call our project "prjWeekOneProgramDoneInClass". Notice that each of the "words" in the above name start with a capital letter, thus following the i-capping rule. Prjweekoneprogramonedoneinclass would be perfectly acceptable to VB .NET but it is NOT acceptable within the realms of this course!

Type "prjWeekOneProgramOneDoneInClass" into the text box alongside the prompt 'Name":



Notice that the "Solution Name" text box defaults to the same name as the project name. You can change this if you wish but it is good practice to leave it the same name. Notice also that there is a check box (ready-checked) that will allow VB .NET to create a directory for the solution. Leave that checked. Do not check 'Add to source control'.

Click on the 'Save' button. Once you have done this you can see that your project has assumed its new name by looking in the top left of the .NET editor screen, as shown to the right.

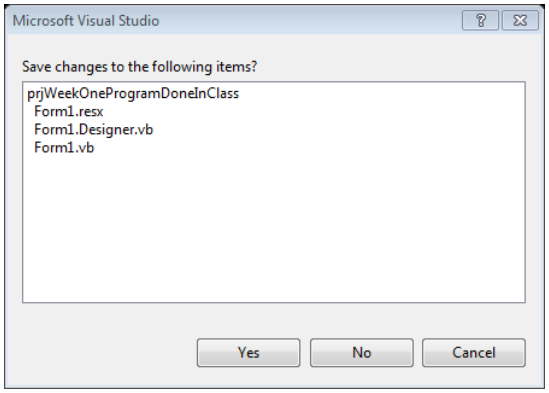


Now you have saved your project for the first time, you need only click on the 'Save' or 'Save All' icons in the icon bar. 'Save' will save the current environment. 'Save All' will save all aspects of the project, including other forms.

Closing a project

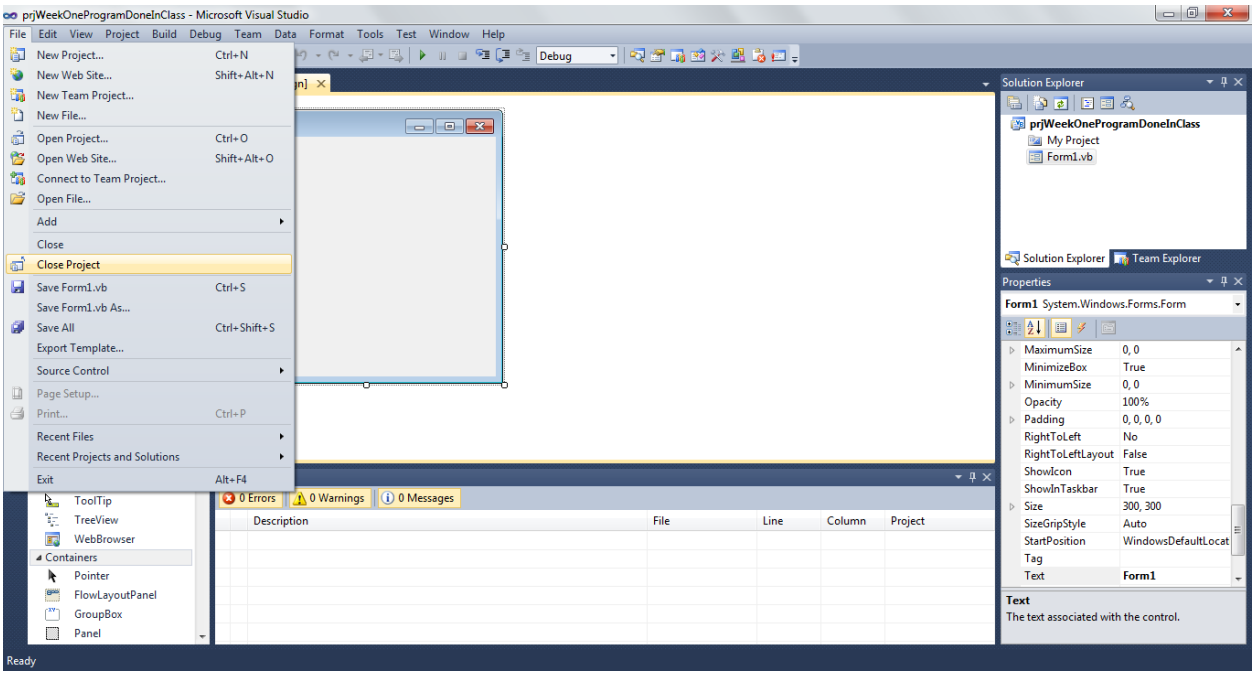
Lecturer: Jane Fletcher

Closing your project is extremely easy; all you do is click on the cross in the top left-hand corner of the .NET environment.

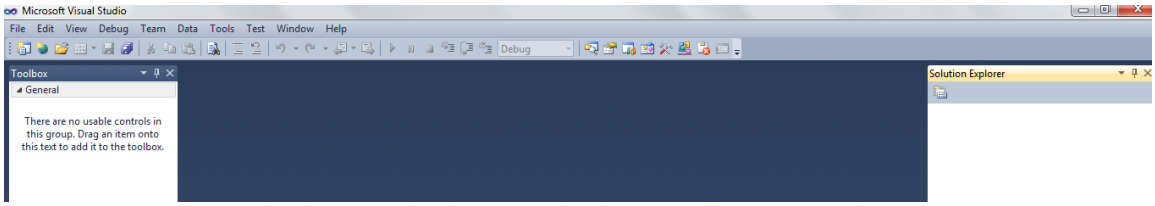


If you have made a change in either design view or coding view of any of the forms within the project and haven't saved since, you will be told by VB .NET after you have clicked the 'close' button that you haven't, and asked if you would like to save, as shown in the illustration to the right. If you would like to quit without saving then you should click the 'No' button. All changes made since your last save will be lost if you do this so be aware! If you want to save changes made to your project, click on the 'Yes' button. If you clicked 'Close' by accident, click on the 'Cancel' button and you will be taken back to the .NET environment, to the place you were at before you clicked on 'Close'.

To close a project without closing VB .NET, go to File in the menu bar, and click 'Close Project'.



Again. if you've made any changes to the project and not saved, you'll be prompted as above. If you close the project you will be in the .NET environment with no active project. In the illustration below you can see that the Solution Explorer view on the right-hand side of the screen is empty. There is no active project.



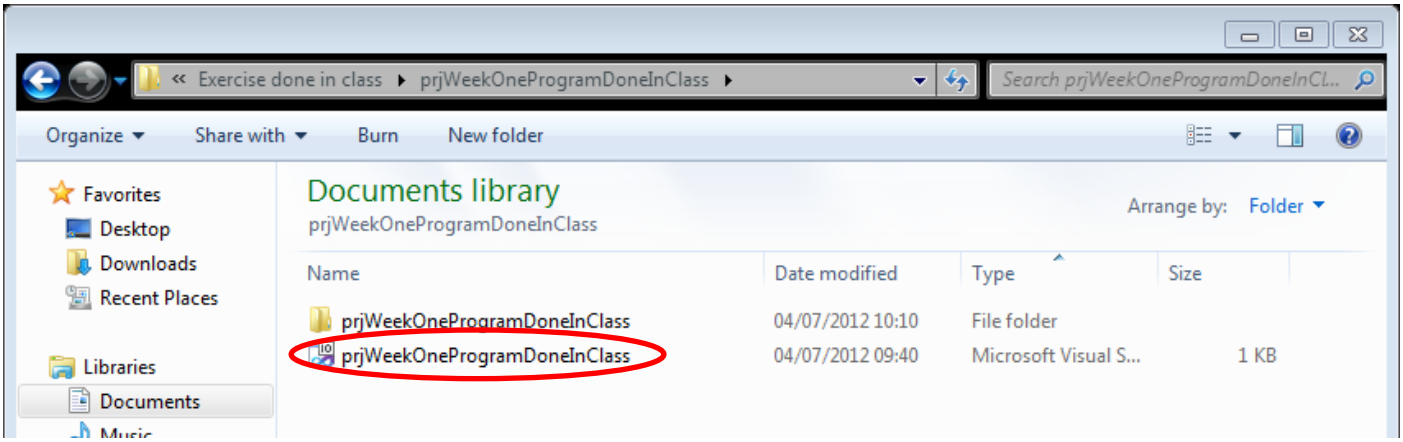
Loading an existing project

Lecturer: Jane Fletcher

There are several ways to load an existing project:

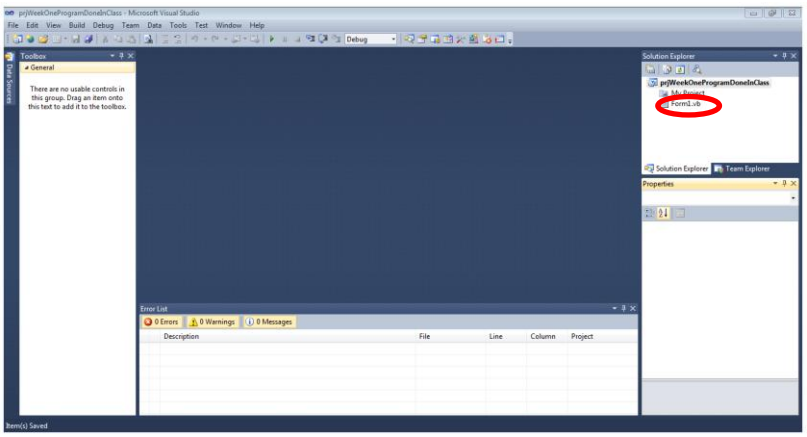
- From Windows Explorer, by clicking on the 'Solution' file of the project;
- By opening Visual Basic .NET and using the 'Recent Projects' list on the Start page;
- By opening Visual Basic .NET and clicking on 'Open Project' on the Start page;
- By opening Visual Basic .NET and clicking File on the menu bar, then 'Open Project'.

Loading a project from Windows Explorer
This is the personal favourite of the author of this document. All you need to do is to open Windows Explorer and navigate to the folder holding your project:

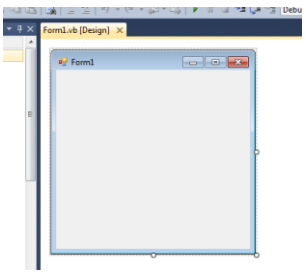


You are looking for the file that is the 'Solution' aspect of the project you have previously created. In our example above, it is the file that is circled. Just double-click on that file. Visual Studio will open into the VB .NET editor for you, and load all aspects of your project. If you already have Visual Studio open, it will open another instance of it for you – it will not overwrite any project that you already have open.

Do not panic if you can't see your form. Visual Studio may open the project for you, but it doesn't always open it in the same view that you closed it. For example, when we open the project created in the 'saving' exercise, prjWeekOneProgramDoneInClass, this is the view we get of the project. Where has my form gone?



You should move across to the Solution Explorer window on the right-hand side of the form, and double-click on 'Form1'. Lo and behold, your form will appear in the central pane.

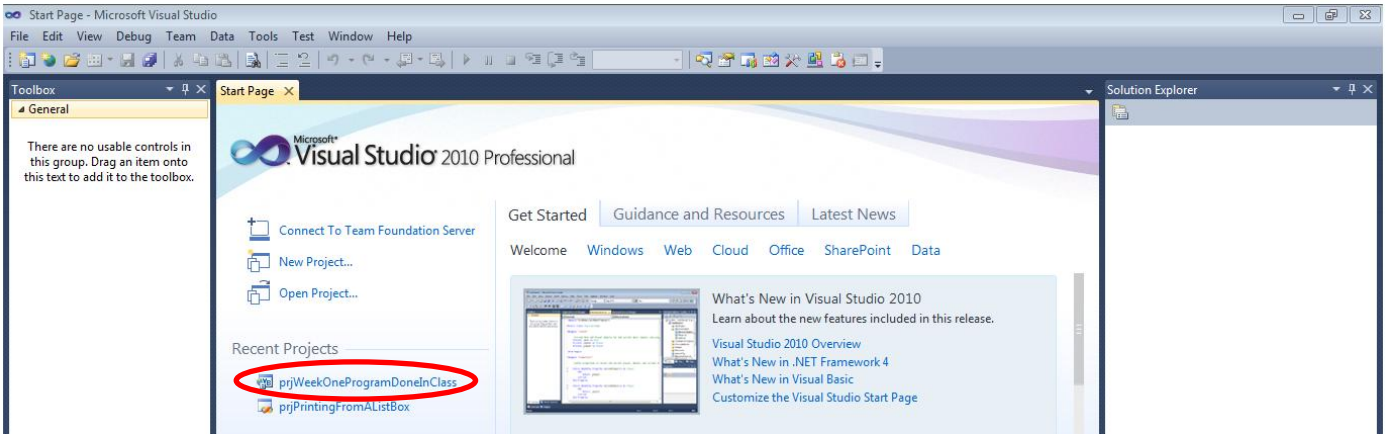


Using the 'Recent Projects' list

This is this author's least favourite method of opening a project – it isn't reliable since it doesn't show the path of where the project is stored. For example, if you decide that you want to save your project on your pen drive to take home, having created it on the College network and saved it on your Z drive, you won't know which occurrence of the project is actually being indicated in the list contained in 'Recent Projects'.

However, assuming that you only have one incidence of each of your projects, we will proceed with opening a project from the 'Recent Projects' list.

Open Visual Studio .NET, using the procedure outlined in pages 54 and 56. The software will load the Start Page for you, and your 'Recent Projects' list will be populated. Notice in the illustration below that the list has now grown and at the top the project we created in the previous sections appears at the top of the list. The most recent project accessed will always be at the top – prjWeekOneProgramDoneInClass in our example here.

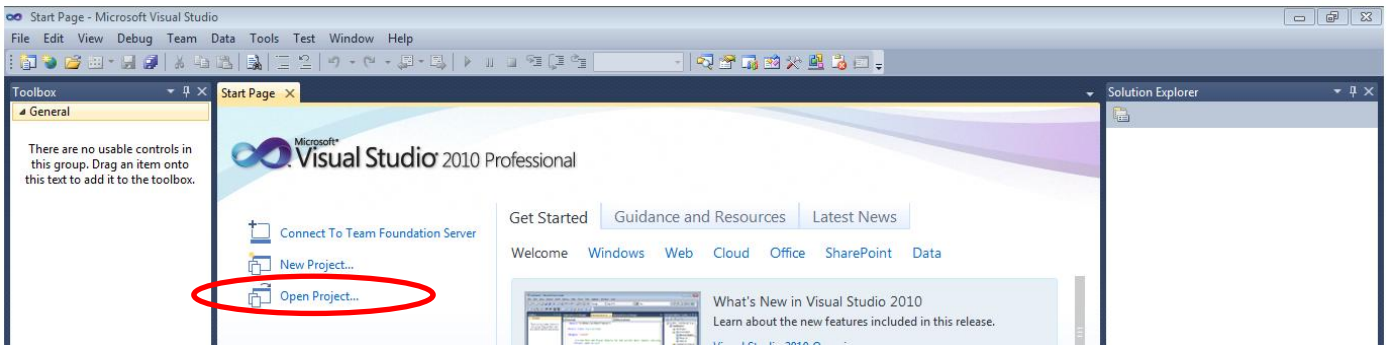


Double-click on the solution file prjWeekOneProgramDoneInClass.

Your project will now open. If you can't see your form, follow the procedure described on page 61.

Using 'Open Project' on the Start Page

Open Visual Studio .NET and on the Start page, click on 'Open Project'.

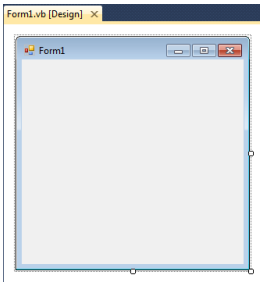


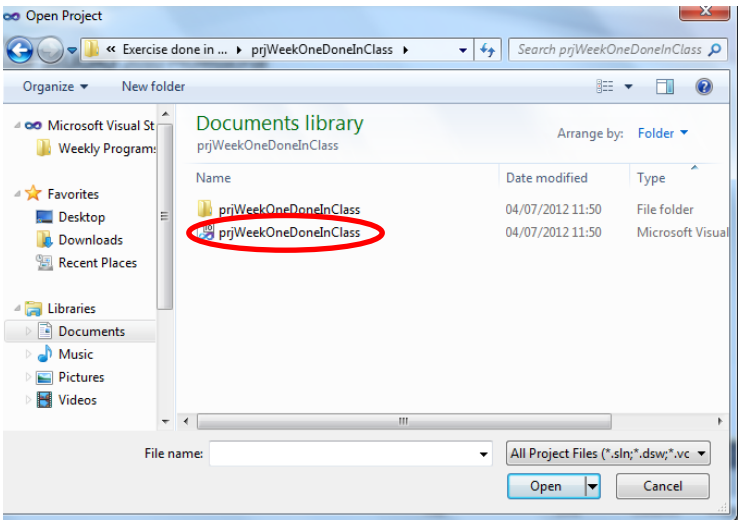
Loading an existing project

Lecturer: Jane Fletcher

Find the 'Solution' file belonging to the project that you want to open and click on it, then click on the 'Open' button.

Alternatively you can double-click on the 'Solution' file and the project will open.

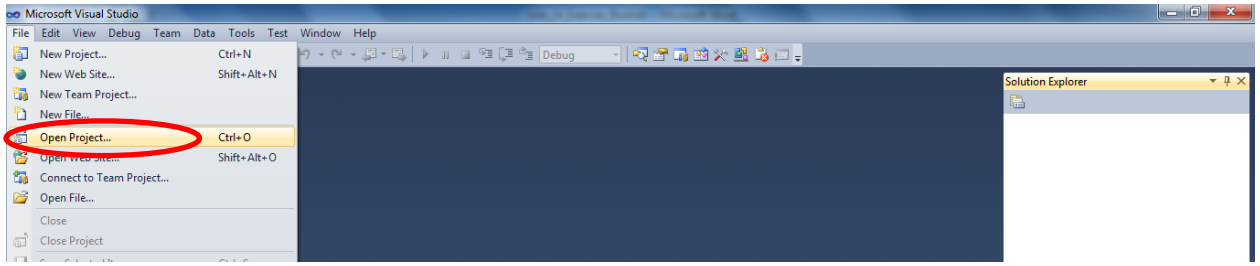




If you can't see your form, follow the procedure outlined on page 51.

Using 'File' and 'Open Project' on the menu bar

Click on 'File' on the menu, then move down to 'Open Project'. Click on that. Navigate your way through the 'Open Project' dialog box until you come to the 'Solutions' file of your project, and either double-click on that file, or click once and then click on 'Open'. Again, if you can't see your form, follow the procedure outlined on page 61.



Data types

Lecturer: Jane Fletcher

A 'data type' is the technical term used within a programming environment for types of information that the programming language is capable of understanding. Each different data type has its own characteristics and validation rules.

Visual Studio .NET supports a range of data types. The table below is taken from the Microsoft website and is a list of data types that VB acknowledges.

Visual Basic type	Common language runtime type structure	Nominal storage allocation	Value range
Boolean	Boolean	Depends on implementing platform	**True** or **False**
Byte	Byte	1 byte	0 through 255 (unsigned)
Char (single character)	Char	2 bytes	0 through 65535 (unsigned)
Date	DateTime	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) † with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28) †
Double (double-precision floating-point)	Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324 † for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 † for positive values
Integer	Int32	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long (long integer)	Int64	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18 †) (signed)
Object	Object (class)	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type **Object**
SByte	SByte	1 byte	-128 through 127 (signed)
Short (short integer)	Int16	2 bytes	-32,768 through 32,767 (signed)

Data types

Lecturer: Jane Fletcher

Visual Basic type	Common language runtime type structure	Nominal storage allocation	Value range
Single (single-precision floating-point)	Single	4 bytes	-3.4028235E+38 through -1.401298E-45 † for negative values; 1.401298E-45 through 3.4028235E+38 † for positive values
String (variable-length)	String (class)	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	UInt32	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	UInt64	8 bytes	0 through 18,446,744,073,709,551,615 (1.8...E+19 †) (unsigned)
User-Defined (structure)	(inherits from ValueType)	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	UInt16	2 bytes	0 through 65,535 (unsigned)

† In *scientific notation*, "E" refers to a power of 10. So 3.56E+2 signifies 3.56 x 10^2 or 356, and 3.56E-2

Reference taken from http://msdn.microsoft.com/en-us/library/47zceaw7(v=vs.80).aspx

Arithmetic and logical comparisons can be performed within any computer program, as long as the appropriate data type is used (see section 'Data types').

Arithmetic operators

The table below shows the arithmetic operators, and also their order of precedence.

Operator	Visual Basic symbol	Order of precedence
Addition	+	6=
Subtraction	-	6=
Multiplication	*	3=
Division	/	3=
Integer division	\	4
Modulus	Mod	5
Negation	-	2
Exponentiation	^	1

Logical operators

There are several logical operators that you will use frequently during this unit. These simple operators are detailed in the table below. There are also some complex logical operators, which are discussed below the table.

Logical Operator	Visual Basic symbol	Example
Equal to	=	If A = B Then
Not equal to	<>	If A <> B Then
Greater than	>	If A > B Then
Less than	<	If A < B Then
Greater than or equal to	>=	If A >= B Then
Less than or equal to	<=	If A <= B Then

More complex logical operators

Logical operators compare Boolean expressions and return a Boolean result. The **And, Or, AndAlso, OrElse** and **Xor** operators are binary because they take two operands, while the **Not** operator is unary because it takes a single operand. Some of these operators can also perform bitwise logical operations on integral values.

And

The **And** operator performs logical conjunction on two Boolean expressions. If both expressions are True, then And returns True. If at least one of the expressions evaluates to False, then And returns False.

```
If (txtFirstNumber.Text < 10) And (txtSecondNumber.Text < 10) Then
    lblOutcome.Text = "Both inputs are less than 10."
End If
```

If the user enters 10 into the first text box (txtFirstNumber) and 5 into the second text box (txtSecondNumber) then the first condition is not met and so the And operator will return False. If the user enters 5 into both text boxes, the And operator will return True.

Or

The **Or** operator performs logical disjunction or inclusion on two Boolean expressions. If either expression evaluates to True or both evaluate to True, then Or returns True. If neither expression evaluates to True, Or returns False.

```
If (txtFirstNumber.Text < 10) Or (txtSecondNumber.Text < 10) Then
    lblOutcome.Text = "One input is less than 10."
End If
```

If the user enters 5 into txtFirstNumber and 10 into txtSecondNumber, the Or operator returns True because one of the conditions is met. If the user enters 5 into both text boxes, the Or operator returns True because *at least one* of the conditions is met. If the user enters 10 into both text boxes, the Or operator will return False because neither condition is met.

Xor

The **Xor** operator performs logical exclusion on two Boolean expressions. If exactly one expression evaluates to True but not both, Xor returns True. If both expressions evaluate to True or both evaluate to False, Xor returns False.

```
If (txtFirstNumber.Text < 10) Xor (txtSecondNumber.Text < 10) Then
    lblOutcome.Text = "One input is less than 10."
End If
```

If the user enters 5 into both text boxes, then both conditions are met so the Xor operator returns False (since only one can be true to meet the criteria of the Xor). If the user enters 10 into both text boxes, then neither conditions are met so the Xor operator returns False. If the user enters 5 into txtSecondNumber and 10 into txtFirstNumber (or the other way around) then the Xor operator returns True because only one of the conditions were met.

AndAlso

The **AndAlso** operator is very similar to the And operator in that it also performs logical conjunction on two Boolean expressions. The key difference between the two is that AndAlso exhibits *short-circuiting* behaviour. If the first expression in and AndAlso expression evaluates to False then the second expression is not evaluated because it cannot alter the final result, and AndAlso returns False.

```
If (txtFirstNumber.Text < 10) AndAlso (txtSecondNumber.Text < 10) Then
    lblOutcome.Text = "One input is less than 10."
End If
```

If the user enters 9 into both text boxes, then the AndAlso operator will return True because both conditions are met. If the user enters 10 into txtFirstNumber and 9 into txtSecondNumber the AndAlso operator will return False, like And. However, if the user enters 10 into txtFirstNumber it doesn't matter what is entered into txtSecondNumber because the first condition wasn't met so the second one won't be evaluated.

OrElse

The **OrElse** operator performs short-circuiting logical disjunction on two Boolean expressions. If the first expression in an OrElse expression evaluates to True then the second expression is not evaluated because it cannot alter the final result, and OrElse returns True

```
If (txtFirstNumber.Text < 10) OrElse (txtSecondNumber.Text < 10) Then
    lblOutcome.Text = "One input is less than 10."
End If
```

If the user enters 5 into txtFirstNumber and 10 into txtSecondNumber, OrElse will only evaluate the first condition – it won't bother with the second one because the first one has fulfilled the OrElse – and will return True. If the user enters 10 into txtFirstNumber and 5 into txtSecondNumber, OrElse will return True, but will evaluate both conditions since the first one doesn't return True. If the user enters 10 into both text boxes then OrElse will return False.

A control structure in its most basic sense is a block of programming code, but more specifically they control the flow of logic through the program. A program is a set of instructions and statements that perform certain functions and process information in a set way to provide a set outcome.

There are three different types of control structure: sequence, selection and iteration.

Sequence

In programming, the word 'sequence' means exactly the same as it does in any other context. It means that the instructions, or actions, or programming statements, always occur and always occur in the given order. No statement is ever missed out and every statement is always performed.

When you write program code one line after another with no choices or loops within it, it is a sequence. The code below is an excerpt from the Click event of btnDisplay and is a simple one written to display a message in a certain format in a label called lblMessage.

```
'This procedure displays a message to the user in lblMessage.

'Set the text of the label to the message you want to use.
lblMessage.Text = "Welcome to Visual Basic .NET!"

'Set the background colour of the label to black.
lblMessage.BackColor = Color.Black

'Set the foreground colour of the label to white.
lblMessage.ForeColor = Color.White

'Make the label visible now it looks right.
lblMessage.Visible = True
```

The green text, each line prefixed by an apostrophe, indicates that this line is a comment to explain the logic, and is ignored by the program compiler. It is there purely to aid understanding of the purpose of the logic for the reader. In this case the statements are very simple and each line doesn't really need a comment but it is good practice to get into the habit of richly commenting your program code. In months to come you will appreciate the effort you made, when you look back at code you have written for an example of how to write certain statements, for example.

The black writing indicates that this text represents either an object within the code (lblMessage, for example) or a reserved word ("=", "black" or "white", for example).

The brown writing indicates that this is a programmer-defined literal, i.e. it is in double quotation marks and whatever appears between these quotation marks is what will show in this case, in the label lblMessage.

These four VB statements will always occur, and will always occur in the order stated above. The only way that these statements will ever occur in a different order is if the programmer decides to move any of the lines of code.

Selection

A selection is a choice. It represents a branch in program logic where a decision is required as to the next course of action, based on condition(s) stipulated by the programmer at design time.

There are several different types of selection constructs, ranging from the simple If .. Then statement to the complex Select Case statement.

If .. Then .. End If

This is the most basic of the Selection construct.

The format of the statement is:

```
If (Comparator1) (LogicalOperator) (Comparator 2) Then
        Actions statements
End If
```

where Comparator1 and Comparator2 could be the values contained within an object (a text box in the example below) or a variable, and LogicalOperator is the method the comparison is to take, such as equal to, greater than, less than etc. (see pages 66 – 68). "Then" is the keyword that pairs with the "If" statement, and "End If" is the key phrase that ends any "If" statement, regardless of the format of that "If" statement (as long as the "If" occurs on more than one line). The Visual Basic editor will automatically indent any action statements that occur between the If .. Then and End If which aids readability. In this context, action statements are lines of VB .NET code that perform whatever is required should the condition stated in the If statement be met.

```
'This procedure determines the highest number from a pair entered by the user.

'Examine the contents of the two text boxes by looking at the text property of each.
If txtFirstNumber.Text > txtSecondNumber.Text Then
    'This means that the first number entered is highest.
    lblOutcome.Text = "First number entered is highest."
End If
```

If .. Then .. Else .. End If

This is the second least complex of the If statement, where if the first condition is not met, then the alternative action statements following the Else are performed. This differs from the If .. Then .. End If since an alternative is offered with the Else that isn't without it.

```
If (Comparator1) (LogicalOperator) (Comparator 2) Then
        Actions statements
Else
        Alternative actions statements
End If
```

The example code below compares the contents of two text boxes and uses the logical operator "greater than". If the contents of the first text box is of a higher value than the

contents of the second box, then the condition is met and the first action statement is performed. If the contents of the first text box is less than *or equal to* the contents of the second text box, then the second action statement is performed.

```
'This procedure determines the highest number from a pair entered by the user.

'Examine the contents of the two text boxes by looking at the text property of each.
If txtFirstNumber.Text > txtSecondNumber.Text Then
    'This means that the first number entered is highest.
    lblOutcome.Text = "First number entered is highest."
Else
    'This means that the first number entered isn't highest.
    lblOutcome.Text = "First number entered is not highest."
End If
```

If .. Then .. ElseIf .. ElseIf .. End If

This is a more complex example of the If statement, and can be used with as many ElseIfs as necessary. It works in exactly the same way as other If statements, but with the additional facility of being able to drill down as specifically as necessary to give a more detailed analysis of the data entered by the user, or the contents of variables etc. When using the ElseIf it is necessary to specify another comparison (it doesn't need to use the same comparators or logical operators as the initial If statement), whereas the Else does not. Once a condition has been fulfilled, the rest of the ElseIfs are ignored and control is passed to the statement after End If.

In the example below, we perform a more detailed analysis of the information input into txtFirstNumber and txtSecondNumber.

```
'Examine the contents of the two text boxes by looking at the text property of each.
If txtFirstNumber.Text > txtSecondNumber.Text Then
    'This means that the first number entered is highest.
    lblOutcome.Text = "First number entered is highest."
ElseIf txtFirstNumber.Text < txtSecondNumber.Text Then
    'This means that the first number entered less than the second number.
    lblOutcome.Text = "First number entered less than the second number."
ElseIf txtFirstNumber.Text = txtSecondNumber.Text Then
    'The contents of both text boxes are the same.
    lblOutcome.Text = "Both numbers are equal in value."
End If
```

Select .. Case .. End Select

The Select .. Case .. End Select statement is another way of testing either the contents of a variable or of an object such as a text box. It can be used when you wish to test for different ranges (as in Example 1 below) or when you have a String (text) variable to examine, as in Example 2). The 'Select Case' part of the statement (the first line) determines which variable's content will be examined, and the individual 'Case' statements following that are the ranges or instances to check for. Once a match has been achieved, the rest of the 'Case' statements are ignored and control passes to the statement following 'End Select'.

It is possible to have a 'Case Else' if you want to have a catch-all clause should none of the 'Case' checks are met.

Example 1

```
'Declare the integer variables to hold the values used within this procedure.
Dim intYearBorn As Integer          'This information is taken from user entry into
                                        'the text box.
Dim intThisYear As Integer          'This information will be the current year, based
                                        'on system date.
Dim intYearDifference As Integer    'This information will be calculated within the
                                        'procedure.

intYearBorn = txtYearBorn.Text      'Get the value entered in the text box into an
                                        'integer variable.
intThisYear = Now.Year              'Take the Year part of the current date / time
                                        'and store.

'Now subtract the year born from the current year and store in the integer variable.
intYearDifference = intThisYear - intYearBorn

'Using Select .. Case, work on the value stored in the integer variable calculated
      'above.
Select Case intYearDifference
    'Use the range aspect of Case to look at the value in intYearDifference
    'and display a message accordingly.
    Case 0 To 11
        lblOutcome.Text = "Age at December 31st will be less than 12."

    Case 12 To 19
        lblOutcome.Text = "Age at December 31st will be less than 20."

    Case 20 To 29
        lblOutcome.Text = "Age at December 31st will be less than 30."

    Case 30 To 39
        lblOutcome.Text = "Age at December 31st will be less than 40."

    Case 40 To 49
        lblOutcome.Text = "Age at December 31st will be less than 50."

    Case Is > 49
        lblOutcome.Text = "Age at December 31st will be 50 or over."

End Select
```

Example 2

```
'The user will enter a grade (P, M or D) into the text box, which will then be
'transferred to the variable chrGrade. This input will then be examined in a
'Select Case and an appropriate message displayed, dependent on value.

Dim chrGrade As Char        'Holds the user's entry after input and assignment.

'Move the value entered by the user, after it has been converted to uppercase.
```

```
chrGrade = StrConv(txtGrade.Text, VbStrConv.Uppercase)

'Using the Select Case statement, output a message depending on the value
'stored within chrGrade.
Select Case chrGrade
    Case "P"
        lblOutcome.Text = "The grade achieved is a Pass."

    Case "M"
        lblOutcome.Text = "The grade achieved is a Merit."

    Case "D"
        lblOutcome.Text = "The grade achieved is a Distinction."

    Case Else
        lblOutcome.Text = "No grade has been achieved."

End Select
```

Iteration

Within the programming context the word 'iteration' can be translated as 'repetition'. There are various types of iteration constructs available, and which one you choose to use will depend on the type of repetition you need:

- You know exactly how many types you want to loop, so you will use For .. Next;
- You want to loop for a certain number of times, once for each occurrence of a particular item in an array for example, so you will use For Each;
- You want to repeat a loop while a certain condition remains true, so you will use Do While .. Loop or Do .. Loop While;
- You want to repeat a loop until a certain condition becomes true, so you will use Do Until .. Loop.

For .. Next

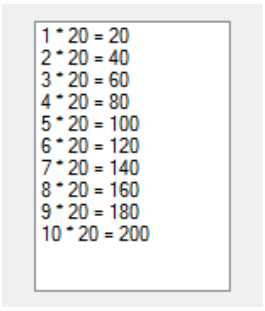
This loop will perform a set of instructions (which are contained within the For .. Next statements) a set number of times, determined by the starting values and ending values which are specified either directly or within variables. With this type of loop you do not have to worry about incrementing any counters because Visual Basic .NET does this for you.

The example below will take two values from the user, one to determine the number to be multiplied, and the other to determine how many times that number will be multiplied, thus generating a "times table" for that number.

```
'This process will take the user's inputs from txtInput and txtMultiplier
'and will use the value in txtInput to determine how many times the
'loop is to be performed, and the value in txtMultiplier is used as
'the figure that will be multiplied. For example, the 12 times table will
'be generated 10 times if the user enters 10 into txtInput and 12 in txtMultiplier.

Dim intCounter As Integer       'The variable used as a counter within the loop.
Dim intMultiplier As Integer    'The number the user wants to multiply.
Dim intAnswer As Integer        'The result of the calculation
```

```
Dim intNumberOfIterations As Integer
'The variable to hold how many iterations the user wants.

'Store the value within txtInput in intNumberOfIterations.
intNumberOfIterations = txtInput.Text

'Store the value within txtMiltiplier in intMultiplier.
intMultiplier = txtMultiplier.Text

'Using intCounter as a, well, counter, loop through the coding within
'the For and Next statements, incrementing intCounter (starting at 1)
'by 1 until intCounter reaches the value stored in intNumberOfIterations.
For intCounter = 1 To intNumberOfIterations
    'Store the value of multiplying the counter with the multiplier.
    intAnswer = intCounter * intMultiplier
    'Output the answer to the list box.
    lstAnswer.Items.Add(intCounter & " * " & intMultiplier & " = " & intAnswer)
Next
```

The Listbox object (lstAnswer) when the user enters 10 into the number of iterations and 20 into the multiplier.



For Each .. Next

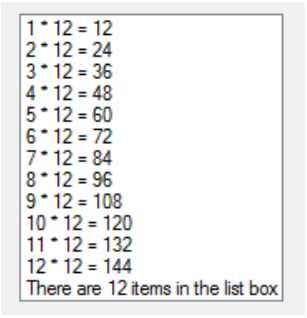
If you have an object (such as a ListBox or ComboBox) or an array of items and want to process each one of them one after the other, use For Each .. Next. The format of this statement is:

For Each *Element* in *Group*
Action statements
Next

where *Element* is the thing in the *Group* that you want to process. In the example below, this coding will follow the coding from the For .. Next example above, and will count how many iterations have been performed in the For .. Next process, using the For Each statement based on the Items in lstAnswer. This is a silly exercise to perform because we know that one of the inputs from the user is exactly that information, but this is used here as a simple example of how to make a For Each .. Next statement work. The output from the program is displayed below the coding.

```
Dim intSumCount As Integer  'Used to count up how many sums have been performed
'and put into the list box.

'Using For Each, add one to intSumCount to count how many items are in the list box.
For Each Item In lstAnswer.Items
    'Add one to intSumCount.
    intSumCount += 1
Next

'Display the counter at the bottom of the list box.
lstAnswer.Items.Add("There are " & intSumCount & " items in the list box.")
```

In the example screenshot here, the user has entered that the number 12 should be multiplied from 1 to 12, and For Each .. Next statement counts how many sums have been performed and displays this information in the last line of the list box.



Do While .. Loop and Do .. Loop While

The Do While .. Loop and Do .. Loop While are very similar. There is, however, a difference between the two – Do .. Loop While will perform the action statements within it at least once, whereas with Do While .. Loop if the condition set after the word 'While' is met already, the loop will not be performed at all. With Do .. Loop While the action statements within the loop will be performed at least once.

Formats for these statements are:

Do While (*condition*)
    Action statements
Loop

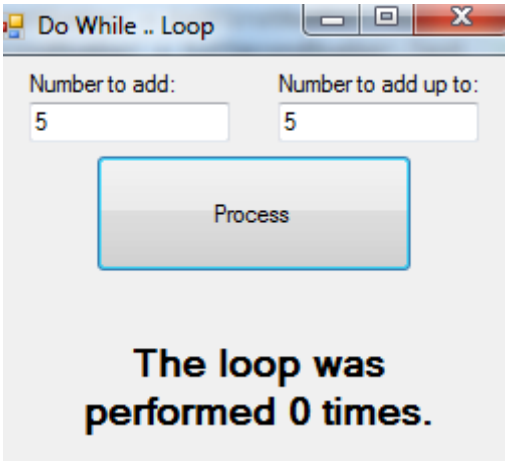
and

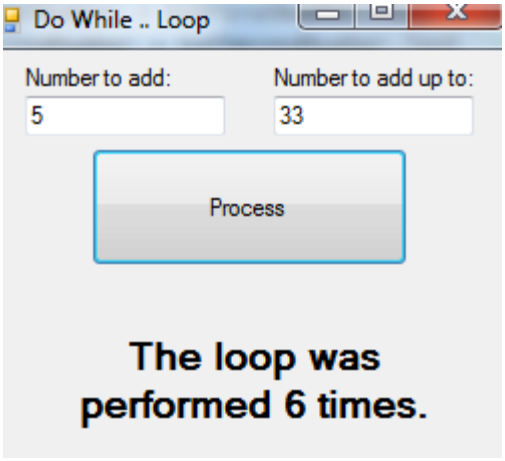
Do
    Action statements
Loop While (*condition*)

In the example below the user is asked to enter two numbers: the first one is the number to add up, and the second one is the total that we mustn't go over. Notice in the first screenshot below the coding that when both numbers are equal, the loop isn't actually performed.

```
'The purpose of this block of code is to take the values from the two text boxes
'entered by the user, and to add the first number to a total (that starts from the first
'number) and keep adding that number to the total until the total is greater than the second
'number entered.

Dim intTotal As Integer
'Used to accummulate all additions performed on the first number.
Dim intFirstNumber, intSecondNumber As Integer
'Used to store the values from txtFirstNumber and txtSecondNumber.
Dim intCounter As Integer   'Used as a counter to see how many times the loop is processed.

intFirstNumber = txtFirstNumber.Text 'Store the input from the user in the integer variable.
intSecondNumber = txtSecondNumber.Text 'This is the value to reach.

intTotal = intFirstNumber 'Set the total to equal the first number straight away.

'The loop will be performed while the accumulated total of first numbers is less
'than the second number entered by the user.
```

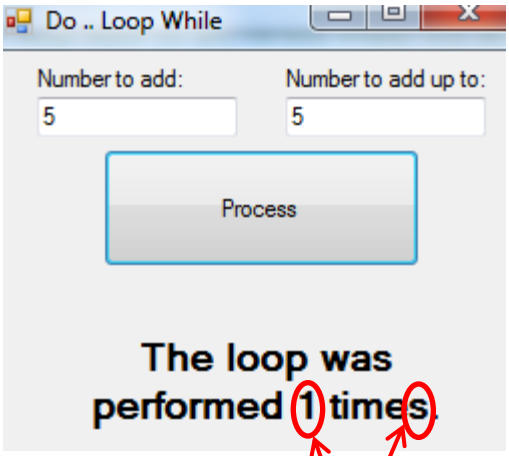
```
Do While (intTotal < intSecondNumber)
    intTotal += intFirstNumber 'Add intFirstNumber to the running total.
    intCounter += 1 'Add one to the counter.
Loop

'Display the outcome in the label.
lblOutcome.Text = "The loop was performed " & intCounter & " times."
```

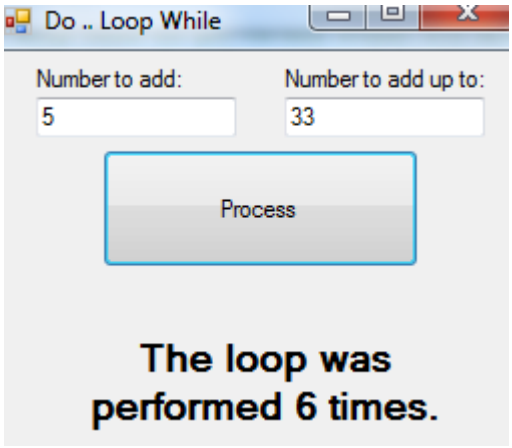




In the coding below, the Do While .. Loop (italicised in the coding above) is replaced by a Do .. Loop While loop, with the rest of the coding remaining exactly the same. The same numbers will be used in the screenshots below - notice the difference!

```
Do
    intTotal += intFirstNumber
    intCounter += 1
Loop While (intTotal < intSecondNumber)
```



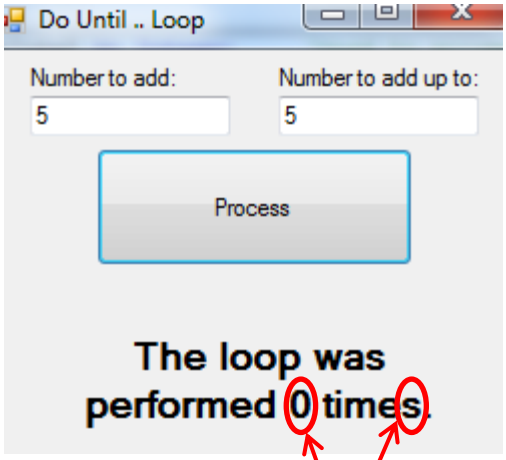


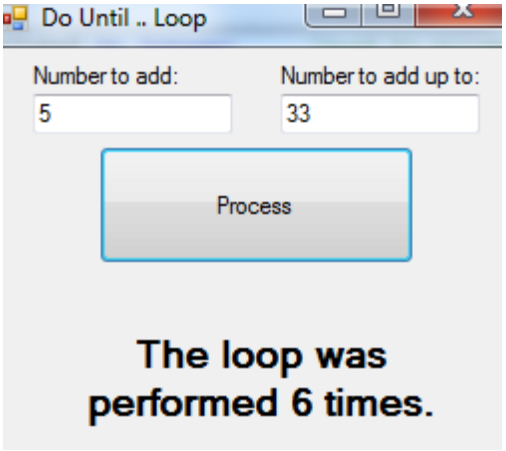
Do Until .. Loop

Instead of specifying the condition as being the current situation (i.e. while it still applies), with the Do Until .. Loop iteration we specify the condition when we will stop, i.e. the future condition. If the condition applies already, because the check is at the top of the loop if the condition already applies then the loop won't be performed at all (like Do While .. Loop, rather than Do .. Loop While).

Again we have used the coding from pages 75 – 76 and have substituted the code on pages 76 that appears in italics with the code below. Notice that the condition has changed here from having the second number being less than the total to the total being greater than or equal to the second number.

```
Do Until (intTotal >= intSecondNumber)
    intTotal += intFirstNumber
    intCounter += 1
Loop
```





Validation techniques

Lecturer: Jane Fletcher

When you are creating your projects initially you will pay lots of attention to the cosmetic appearance of your work; you will experiment with colour, font, line-up and layout – all this is completely proper and what you should be doing. However, when you get further on in your development of software it will become important that you ensure that all information that is entered by the user into your projects is correct.

There are different ways of ensuring validation is thorough when you are writing your projects. The first way is to ensure that you use the appropriate data types when collecting data from an input object (such as a TextBox) and assigning it into a variable. We have already explained about the different data types that are available to you when you are assigning data; the first thing you need to consider is whether the data is numeric or non-numeric. If the data is numeric, then you have further decisions to make because you have to work out what type of numeric data it is. For example, is the data integer (whole number), decimal, a small number with decimal places, a really large number with lots of decimal places and so on. You also have to take into account what precision is required; if you are calculating Pi to 1,000 decimal places then this will require more precision than if you are adding up the cost of three CDs that you're buying in a shop.

When you are writing your validation code these are the things you must consider:

- Whether there is anything entered in the first place;
- Whether the thing that is entered is too long;
- Whether the thing that is entered is of the correct type (this applies to numeric data);
- Whether the thing that is entered is within the correct range (this applies to numeric data).

There are more things to consider when validating user input, but the above items are the most important.

The coding below is written to ensure that the user has entered a numeric value in the range of 99p to £99.99 into the TextBox object txtInput and belongs to the Click event of btnValidate.

```
Private Sub btnValidate_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles btnValidate.Click

    'Check that the user has actually entered something, and display a
    'message to the user if not.
    If txtInput.Text.Length = 0 Then
        MsgBox("You must enter the price of the item being purchased.", _
                MsgBoxStyle.Critical, "NO INPUT!")
        txtInput.Focus()    'Put the cursor back into the text box.
        Exit Sub            'Quit this procedure because user entry is invalid and we
                              'can't proceed.
    End If

    'Check that the user han't entered too much input. Display a message if they have.
    If txtInput.Text.Length > 5 Then
        MsgBox("You must enter a price in the range of 99p to 99.99.", _
                MsgBoxStyle.Critical, "TOO MUCH INPUT")
        txtInput.Text = ""  'Erase invalid text from the text box.
```

```
            txtInput.Focus()    'Put the cursor back into the text box.
            Exit Sub            'Quit this procedure because user entry is invalid and we
                                  'can't proceed.
        End If

        'Check that the entry made by the user is numeric. If it isn't,
              'display a message to the user.
        If Not IsNumeric(txtInput.Text) Then
            MsgBox("You must enter a price in the range of 99p to 99.99.", _
                    MsgBoxStyle.Critical, "NON-NUMERIC INPUT")
            txtInput.Text = ""  'Erase invalid text from the text box.
            txtInput.Focus()    'Put the cursor back into the text box.
            Exit Sub            'Quit this procedure because user entry is invalid and we
                                  'can't proceed.
        End If

        Dim decPrice As Decimal 'This is the variable to hold the input from the text box.

        decPrice = txtInput.Text 'Assign the value from the text box into the numeric
                                  '(decimal) variable.

        'Check that the information in the variable is within the range of 99p to £99.99.
        If (decPrice < 0.99) Or (decPrice > 99.99) Then
            MsgBox("You must enter a price in the range of 99p to 99.99.", _
                    MsgBoxStyle.Critical, "INPUT OUT OF RANGE")
            txtInput.Text = ""  'Erase invalid text from the text box.
            txtInput.Focus()    'Put the cursor back into the text box.
            Exit Sub            'Quit this procedure because user entry is invalid and we
                                  'can't proceed.
        End If
    End Sub
```

Notice that the user isn't supposed to enter either a 'p' for pence, or a pound sign. This would make the input be classed as text rather than numeric.

If you are validating the contents of a TextBox object for non-numeric input, then you would just need to use the first two checks: for length being = 0, meaning that the user hasn't entered anything; and length being greater than a set number of characters.

There are many more complicated ways of validation which you could use but are for you to research and try out, rather than for the author to explain here.

Making a project look more professional

Lecturer: Jane Fletcher

By the time that you have got into the second semester of this course you should be aiming towards producing projects that look, and to work, in a professional way. This means having no bugs within the logic, ensuring that the tab order is correct on each form, and appropriate help facilities and error messaging available to assist the user in the operation of the software.

What do you notice when you load any Microsoft software, such as Word, Excel, PowerPoint, Access, Visual Studio etc? Each of the individual parts of the Office suite have their own individual start-up screen, or as it is known in the trade, splash screen. The idea behind the splash screen is to tell the user what the software is that is loading, and to give the user something to look at while the software itself loads.

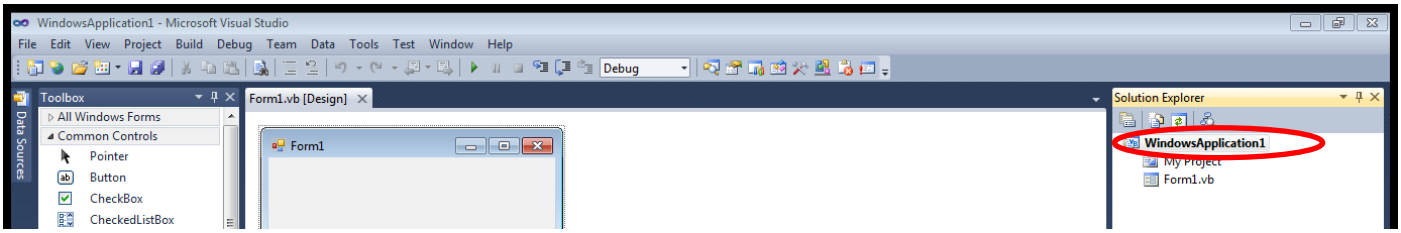
Creating splash screens

There is a template splash screen available to the developer once a new project has been started. However, there is a considerable amount of coding attached to this screen that you may not want to use, so be prepared to edit heavily when you're in the coding view.

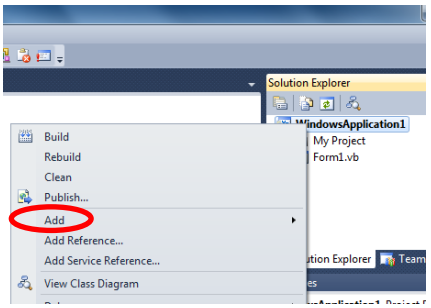
To load a splash screen template there are several methods available but the two easiest ways will be explained here.

Method 1 – using Solution Explorer

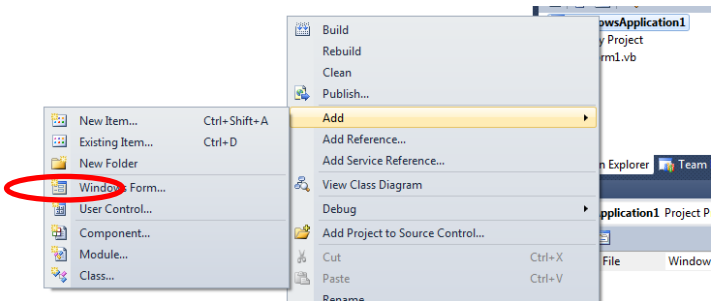
Right-click on the project's name in the Solution Explorer window, shown below.



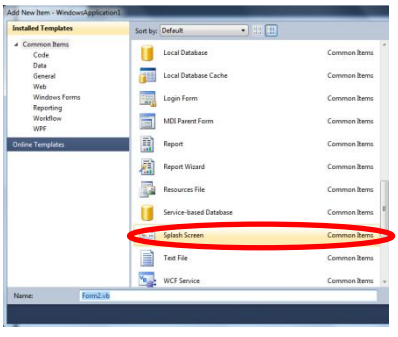
When you have right-clicked, a context menu will appear. Left-click on 'Add'.



A further context menu will appear once you have clicked Add. From this, you should click Windows Form.

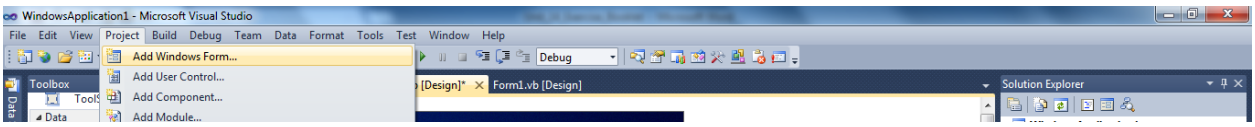


You will be presented at this stage with the Add Form dialog box. Scroll down the list of available templates and click "Splash Screen", then click the "Add" button.

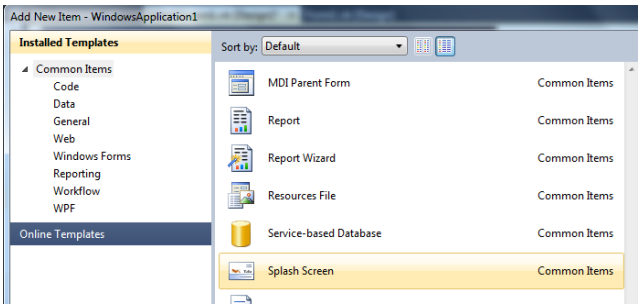


Method 2 – using the Project menu item

An alternative method to add a form to an existing project is to click on Project in the menu bar, and from there click on Add Windows Form.

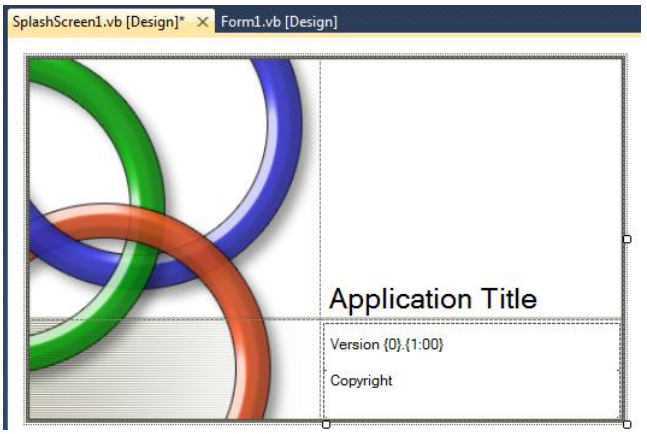


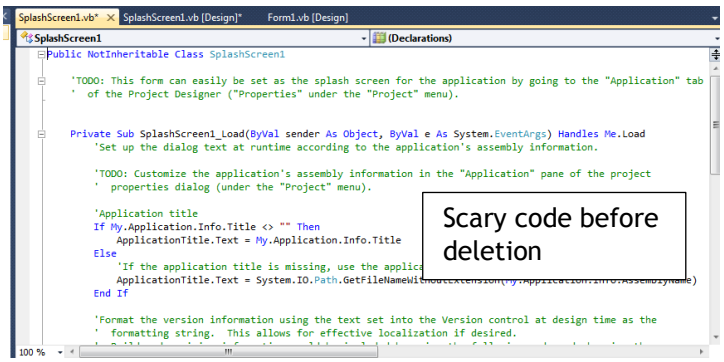
From there select "Splash Screen from the 'Add New Item' dialog box, and then click on the 'Add' button. From there the steps are exactly the same as for Method 1.



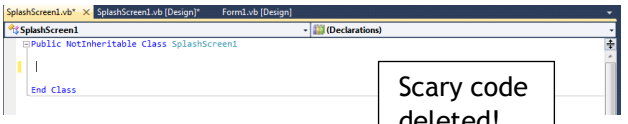
Making a splash screen

The template splash screen generated by Visual Basic .NET will be loaded as an additional form into the current project. It will always have the same appearance, shown here to the right. There is lots of code attached to the template, shown below. You won't want this – so you can safely delete all except the top line and the bottom line (***Public NotInheritable Class SplashScreen1*** and ***End Class***).



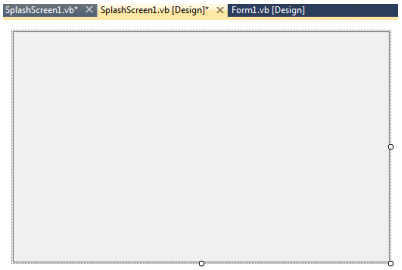


Scary code before deletion



Scary code deleted!

Below is a screenshot of a completely blank splash screen. It has had all its objects removed, including the panel. It is now a blank canvas and you can put whatever image(s), labels, ProgressBars etc that you think is appropriate to your project.

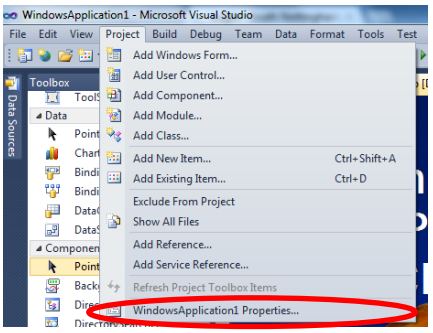


One object that you MUST include, however, is a Timer – this is what will keep your splash screen active and will load the first 'real' screen of your project.

Making a project look more professional

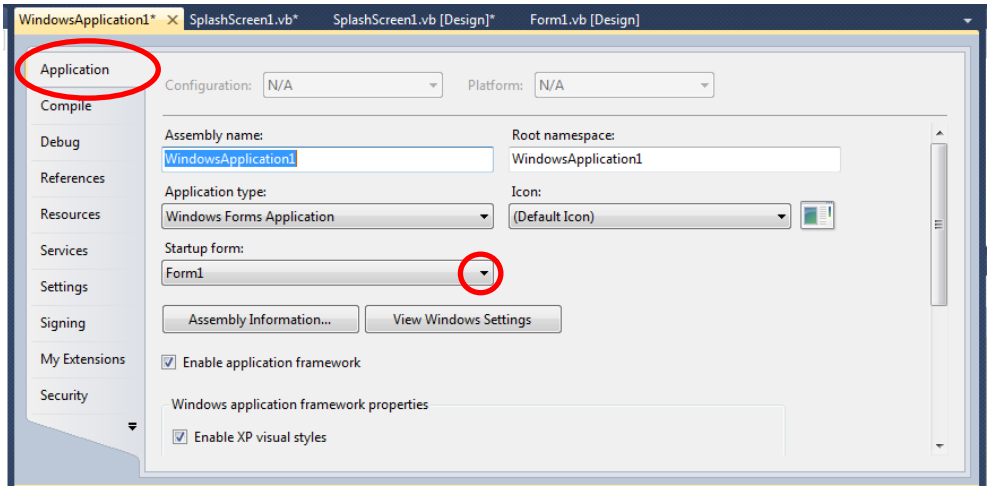
Lecturer: Jane Fletcher

The first thing you are going to have to do is to ensure that the splash screen is the first screen within the project to load; the default loading screen is always Form1 so this must be altered. To do this, click on 'Project' on the menu bar, and then the last item in the list which is always the project's name followed by the word 'Properties'.



In the screenshot here the project hasn't been saved yet so is called 'WindowsApplication1'. The last item in the list of items available from the Project menu is 'WindowsApplication1 Properties'. Click on that.

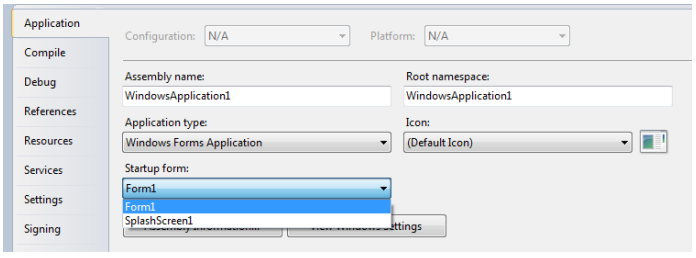
The Properties window for the project will open in the central pane of the editor.



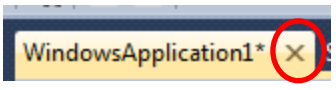
Make sure that the Application tab is the active one by clicking on the appropriate tab.

Notice halfway down the form there is a ListBox underneath the label saying 'Startup form'. Click on the arrowhead to the right of the ListBox.

Once you've clicked on the arrowhead, a list of the forms available within the current project will populate the Items property of that ListBox. In the example to the right you will see that Form1 and SplashScreen1 are available, and that Form1 is selected as the current 'Startup' form. Click on Splashscreen1.



Close the Properties window for the project by clicking on the X button on the Properties top tab.



Here is a shot of the design view of the splash screen that has a background image on the form and a label stating that "This is a SPLASH SCREEN!", and a Timer object visible in the lower pane.

The Interview property for the Timer is set to 3000 (i.e. three seconds) and the Enabled property should be set to True. If you want the splash screen to be visible longer, make the interval longer.

Coding to make the splash screen work is very simple. On the Tick event of Timer1, add coding

```
'Disable the timer so that multiple occurrences of Form1 don't load.
Timer1.Enabled = False
'Declare a new instance of Form1 - you should use the same name as the form here.
Dim Form1 As Form1 = New Form1
'Hide the current form (SplashScreen1).
Me.Hide()
'Show Form1 to the user in place of SplashScreen1.
Form1.ShowDialog()
'Close Form1.
Me.Close()
```

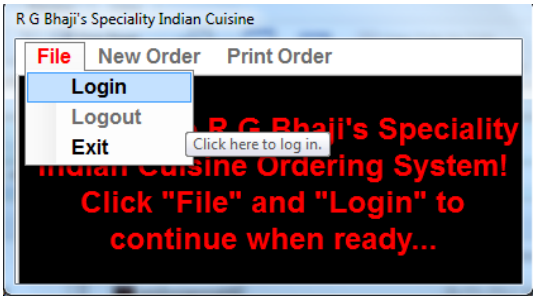
Every programmer develops their preferred method of displaying their splash screens – the above is the author's personal favourite. Play around with the coding above and develop a method you like.

Using the MenuStrip control

One thing that all users of computer software are accustomed to is the MenuStrip. They may not know it as such but they have been conditioned to expect to see "File", "Edit", "View" and so on reading from the left on the top active row of any software they use. Remember the fuss when Microsoft brought out Office 2007 and the familiar menu bar was replaced by the Ribbon? People do not take kindly to change, so bear this in mind when you are writing your programs.

The MenuStrip object (see page 30) is a very easy way to make your project look more professional, as long as you use it responsibly and professionally and stick to commonly used protocol when you design the layout of the menu. Users like the familiar, and hate change – this is a very important point to take into consideration when you're putting together your own software.

Here to the right is an example of a MenuStrip in use in a piece of software created by this author. Note the fact that some items are enabled and some aren't – when the Enabled property is set to False, the item appears greyed out as the 'Logout', 'New Order' and 'Print Order' items are here.



Notice also the fact that the 'Login' option is highlighted. Below it there is a line of text saying 'Click here to log in.' This is another easy way to give your project a more professional appearance: the Tooltip.

Using the Tooltip control

See pages 43 and 44 for an explanation of using the ToolTip object. As explained there it is an excellent way of providing onscreen help for the user, and for partial meeting of the criterion P6 for Unit 14 – Event Driven Programming. Try hovering over the Ribbon in Microsoft Word if you want to see some very well designed usage of the ToolTip object. The key thing is to always be professional in your messages to the user; swearing, joking etc may appear to be a good idea and the height of sophisticated wit when you are in the classroom but neither make users or lecturers smile.

Design documentation

Lecturer: Jane Fletcher

Part of the project development process is the creation of the design documentation that goes with each project that is created. For each of the programs that you will create in class you will be given partial documentation: you will be given a program specification (a detailed breakdown as to how the project should function, including any validation and processing requirements) but for your final assignment you will be expected to create your own.

As a minimum, project documentation should include:

- An overview of the project, explaining briefly its purpose;
- Detail on the inputs to, outputs from and processes in the project; (this may take the form of form designs and dictionaries, data dictionaries, report dictionaries and structured English (see next section);
- Full structured test data, test plans (and latterly, the results of the test plans);
- Full and detailed evaluations of the software you have produced, including why you used the tools and techniques you selected and what you would do differently if you repeated the project, and what you would add if you had more time (future enhancements to the system);
- Full print of commented code;
- Contents page;
- Front cover.

Below is an example of the design documentation (up to the dictionary part) of a project that adds two numbers together and displays a total.

An overview of the project
This section of the documentation requires a brief explanation as to the purpose of the proposed software. An example could be:

"A programmer is required which allows the user to enter two numbers into two separate text boxes. The numbers entered should be integers within the range of 1 to 9999 and any invalid input should be highlighted to the user who should be directed to re-enter the number(s), both on the click event of the button mentioned below.

"Once the numbers have been entered, the user should be allowed to click a button (see previous paragraph) which should validate the user's input. Should the input be valid, then both numbers should be added together and the result stored in an integer variable 'intTotal'. The value in intTotal should then be displayed to the user in lblMessage in the format:

FirstNumberEntered + ***SecondNumberEntered*** = ***intTotal***

"There should be another button available which allows the user to remove the displays and to reset the screen to its original state. This button should clear the label's Text property, the Text properties of both text boxes should be blanked out and the cursor's focus put into the first text box.

"There should be a second button available to allow the user to close the software. Once the button to close has been clicked, a messagebox should be displayed to the user saying 'Goodbye' and the project should close."

Inputs to the project
First integer into the text box used for this purpose (intFirstNumber);
Second integer into the text box used for this purpose (intSecondNumber).

Outputs
Total of both numbers added together, which is to be displayed in the label for that purpose (intTotal).

Processes
Click Event for btnProcess

If txtFirstNumber's text length = 0 Then
Display message to the user stating that an error has occurred
Put the focus of the cursor into txtFirstNumber
Exit the subroutine
End If

If txtFirstNumber's text length > 4 Then
Display message to the user stating that an error has occurred
Remove the text from txtFirstNumber
Put the focus of the cursor into txtFirstNumber
Exit the subroutine
End If

If txtFirstNumber's text is not numeric
Display message to the user stating that an error has occurred
Remove the text from txtFirstNumber
Put the focus of the cursor into txtFirstNumber
Exit the subroutine
End If

Store the value in the Text property of txtFirstNumber in intFirstNumber

If (intFirstNumber < 1) or (intFirstNumber > 9999)
Display message to the user stating that an error has occurred
Remove the text from txtFirstNumber
Put the focus of the cursor into txtFirstNumber
Exit the subroutine
End If

If txtSecondNumber's text length = 0 Then
Display message to the user stating that an error has occurred
Put the focus of the cursor into txtSecondNumber

Exit the subroutine
End If

If txtSecondNumber's text length > 4 Then
Display message to the user stating that an error has occurred
Remove the text from txtSecondNumber
Put the focus of the cursor into txtSecondNumber
Exit the subroutine
End If

If txtSecondNumber's text is not numeric
Display message to the user stating that an error has occurred
Remove the text from txtSecondNumber
Put the focus of the cursor into txtSecondNumber
Exit the subroutine
End If

Store the value in the Text property of txtSecondNumber in intSecondNumber

If (intSecondNumber < 1) or (intSecondNumber > 9999)
Display message to the user stating that an error has occurred
Remove the text from txtSecondNumber
Remove the text from txtSecondNumber
Put the focus of the cursor into txtSecondNumber
Exit the subroutine
End If

Add intFirstNumber and intSecondNumber and store the result in intTotal

Display intTotal in lblMessage's text property in the format
intFirstNumber + intSecondNumber = intTotal
where the figures appear.
End Sub

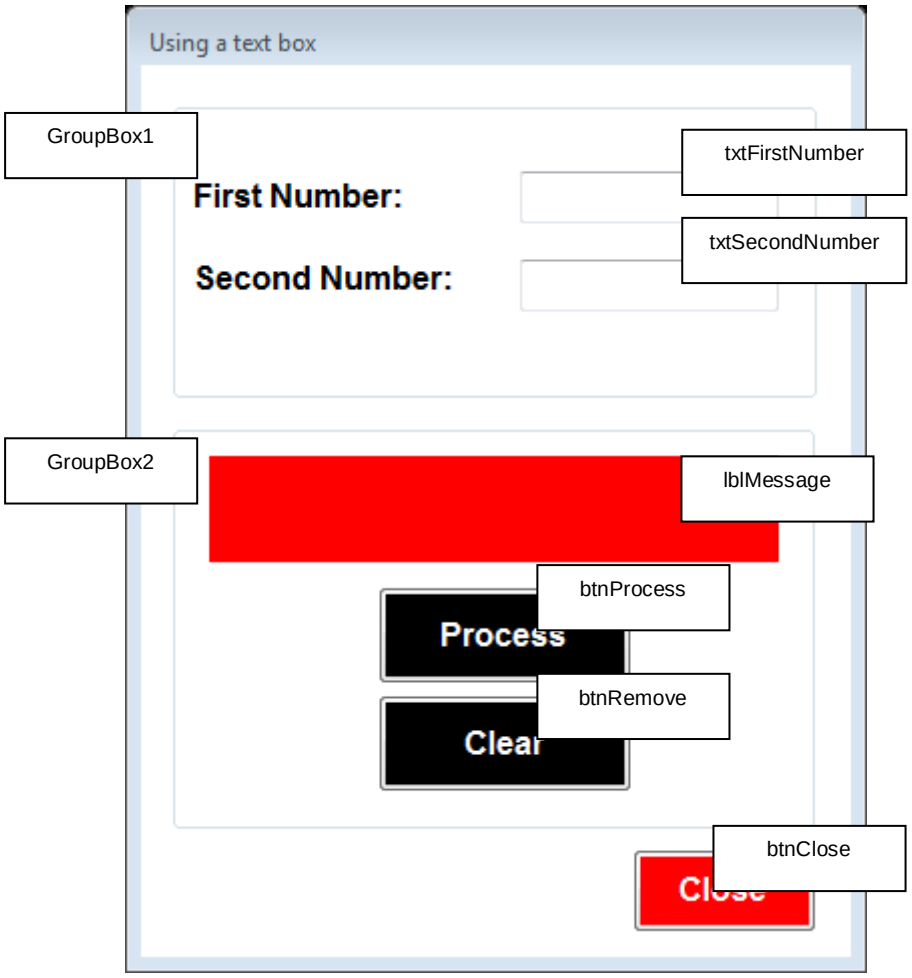
Click Event for btnRemove
Clear out txtFirstNumber's text property
Clear out txtSecondNumber's text property
Clear out the text in lblMessage's text property
Set the cursor's focus in txtFirstNumber
End Sub

Click Event for btnClose
Display a messagebox to the user saying 'Goodbye'
Close the project
End Sub

Design documentation

Lecturer: Jane Fletcher

Form Design



Form Dictionary

Form	Control	Properties	Change to	Used for / scope
Form1	Form1	BackColor	White	User entry of two integers.
		ControlBox	False	Takes away the 'close' feature.
		Font	Arial 12, Bold	
		ForeColor	Black	
		Size	370, 483	
		StartPosition	CentreScreen	
		Text	Using a text box	
	GroupBox1	Location	16, 12	Grouping user input boxes
		Size	321, 155	
	Label1	Location	173, 41	
		Size	129, 26	
		Text	First Number	Prompt to the user
	Label2	Location	173, 85	
		Size	129, 26	
		Text	Second Number	Prompt to the user

	TextBox1	BackColor	White	Input of the first number from the
		ForeColor	Black	user.
		Location	173, 41	
		Name	txtFirstNumber	
		Size	129, 26	
		TabIndex	0	
	TextBox2	BackColor	White	Input of the second number from
		ForeColor	Black	the user.
		Location	173, 85	
		Name	txtSecondNumber	
		Size	129, 26	
		TabIndex	1	
	GroupBox2	Location	16, 173	Grouping together output and
		Size	321, 155	processing buttons.
	Label3	BackColor	Red	Displaying the output message
		Font	Arial 16, Bold	to the user.
		ForeColor	Black	
		Location	18, 22	
		Name	lblMessage	
		Size	284, 53	
		TextAlign	Centre	
	Button1	BackColor	Black	Validates contents of text boxes
		Font	Arial 12, Bold	and displays the total to the user
		ForeColor	White	Assuming all input is correct.
		Location	102, 87	
		Name	btnProcess	
		Size	127, 49	
		Text	Process	
	Button2	BackColor	Black	Removes the contents of text
		Font	Arial 12, Bold	boxes and labels
		ForeColor	White	
		Location	102, 141	
		Name	btnRemove	
		Size	127, 49	
		Text	Remove	
	Button3	BackColor	Red	Closes the project.
		Font	Arial 12, Bold	
		ForeColor	White	
		Location	245, 391	
		Name	btnClose	
		Size	92, 42	
		Text	Close	

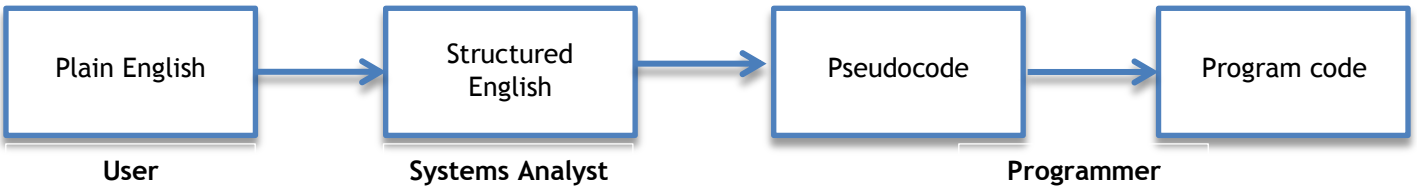
Data Dictionary

Variable name	Type (eg integer, string)	Valid range	Form	Used for
intFirstNumber	Integer	1 – 9999	Form1	Input from txtFirstNumber
intSecondNumber	Integer	1 – 9999	Form1	Input from txtSecondNumber
intTotal	Integer	Calculated	Form1	Calculated by project

Structured English

Lecturer: Jane Fletcher

What is structured English?
Structured English is a language that comes between 'ordinary' English and program code and is written using real English verbs rather than Visual Basic .NET keywords.



In the real world (depending on the size of the company and the number of employees in the IT systems development department) these tasks would be performed by different people.

First of all either the project manager or the systems analyst would work with the user(s) to establish the requirements of the new system. The systems analyst would then write a Requirements Specification, which is the 'plain English' part of the above process. In this document the full specification for the system, including form and report designs, could be laid down and signed off by the user. The user signing off is an extremely important part of the process since it means that you have proof of what the user wants, if it comes to a point later on in the development process when the user claims that he or she asked for something different to what the developers have produced.

The programmer (or developer – the terms are interchangeable) will then take the project specification and from that create either structured diagrams and / or structured English to show step-by-step how the logic of each part of the system will work. The developer then has a structured walk-through of the structured designs with the systems analyst, to get the seal of approval before carrying on to the programming / software building stage.

How do I write structured English?

- Structured English uses a subset of the English language that we write every day. It uses:
  - Verbs ("doing" words) *and*
  - Nouns ("naming" words).
- It does NOT use
  - Adjectives ("describing name" words) *or*
  - Adverbs ("describing doing" words).
- It should read like ordinary English.
- It should have appropriate indentation.

Tips for writing clear structured English

- There are no set standards for structured English.
- However, keep your language "tight" – don't use description, for example.
- Write in blocks, and use good indentation to make it easy to identify what code belongs to which process.
- Keep to the sequence in which events happen.
- Identify choices (selections – "Ifs" and "Select Case"; iterations – loops for how many times you will do something).

Writing structured English

- Verbs to use when writing structured English:

Get	**Read**	**Accept**
Write	**Print**	**Sort**
Move	**Merge**	**Add**
Divide	**Subtract**	**Multiply**
Display	**Calculate**	

- Writing nouns: use i-capping!
- It's legitimate to use variable names in structured English.

Writing iterative statements in structured English

FOR ***condition***
    Iterative statements
NEXT

Or

DO WHILE ***condition***
    Iterative statements
ENDDO

Or

DO
    Iterative statements
UNTIL ***condition***

Notice the indentation here – the iterative statements between the looping clauses are indented from the looping clause, and each looping clause start has to have an end looping clause, which must line up.

Writing selection statements in structured English

IF ***condition***
    Imperative statements
END IF

Or

```
SELECT CASE Item
        CASE Item condition
                Statement(s)
        CASE Item condition
                Statement(s)
        CASE Item condition
                Statement(s)
END SELECT
```

Where does structured English fit in?

- Structured English is written AFTER the design of a program (form designs, form dictionaries, data dictionaries); *but*
- BEFORE the program itself!

Why do it? Do I have to?
YES! You have to do it

- Because it helps you to determine logical order for your statements;
- Because it helps you to determine all the conditions to be applied in your program;
- Because it helps you get P3 on the grading grid!

There is an example of structured English in the previous section, 'Design documentation'. Also there is a full example of a specification and the associated structured English over the page.

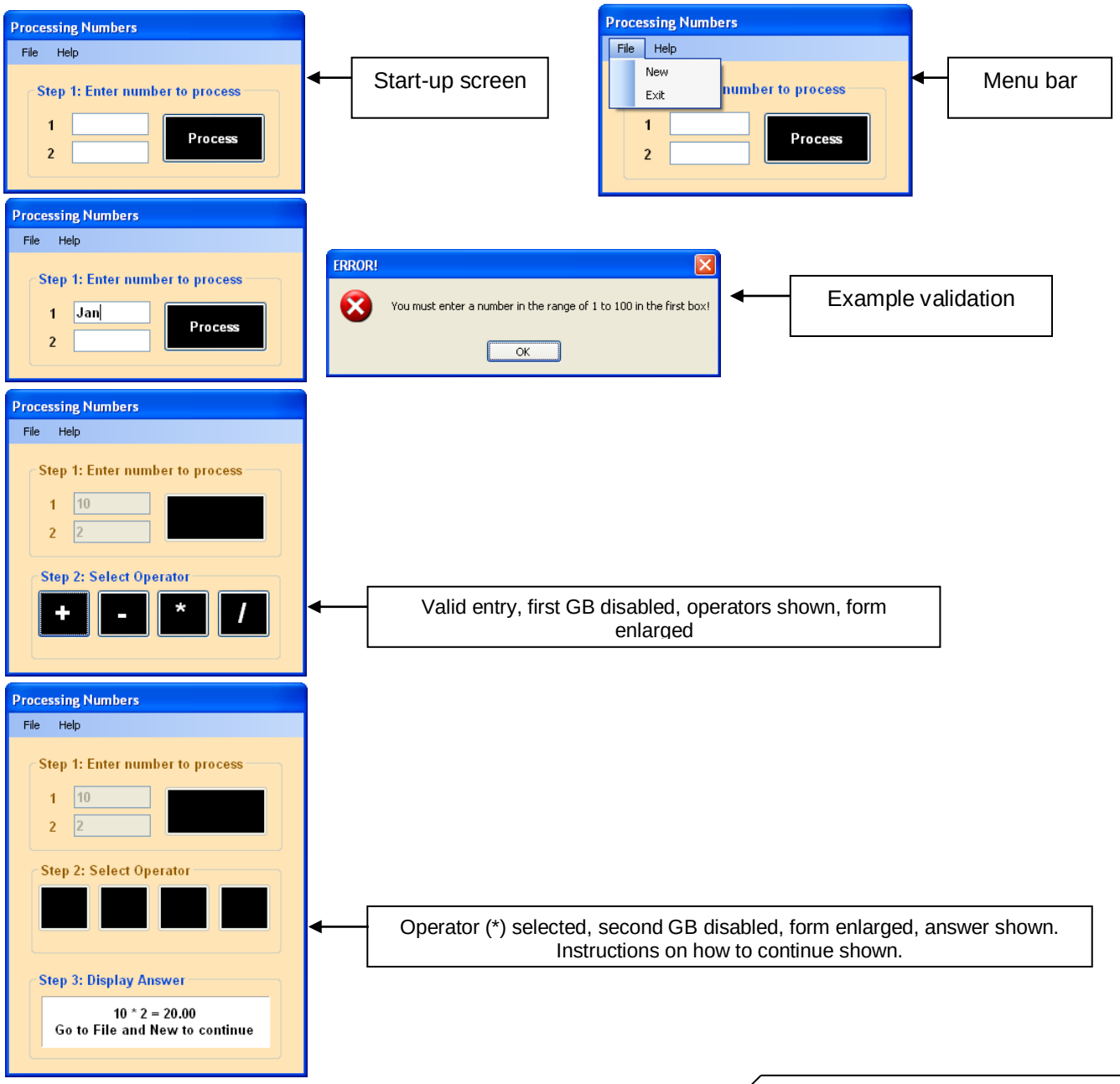
Program specification

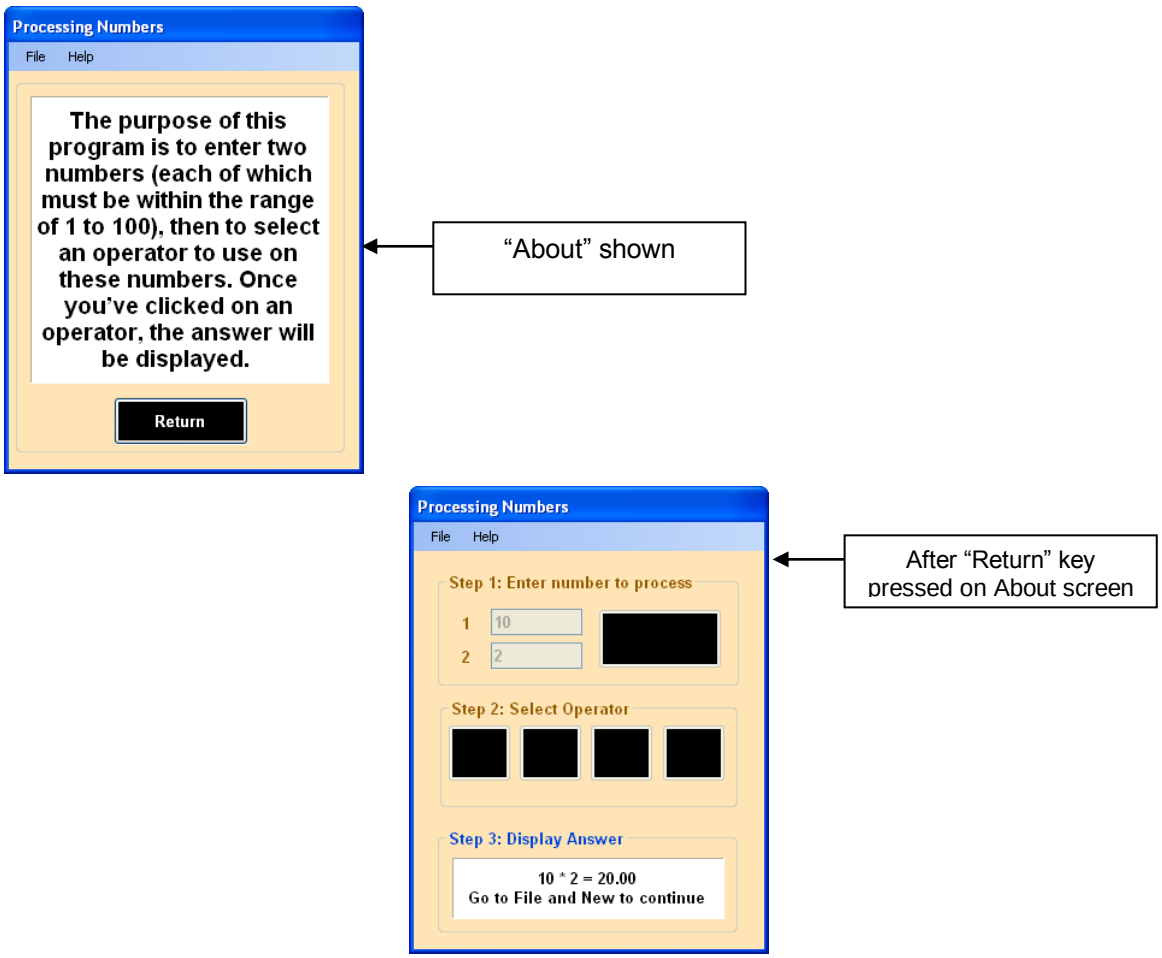
A program is required which will allow the user to enter two numbers, each of which should be within the range of 1 to 100. After the numbers have been entered and proved valid, the user should be allowed to select a mathematical operator (+, -, * or /) and the resulting calculation should be displayed.

An "About" feature is also required to tell the user how the program functions.

The program should be internally documented in full.

Form designs





Structured English

Form_Load

Set the height of Form1 to be 182
Set focus to txtNumber1

End of Form_Load

---

btnProcess_Click

If the length of the entry in txtNumber1 = 0 Then
    Display a message box saying "You must enter a number in the first box!"
    Set the cursor focus to txtNumber1
    Quit the subroutine
End If

If the length of the entry in txtNumber1 is > 3 Then

```
        Display a messagebox saying "You must enter a number in the range of 1 to 100 in
        the first box”
        Blank out the text in txtNumber1
        Set the cursor focus to txtNumber1
        Quit the subroutine
End If

If the entry in txtNumber1 isn’t a number
        Display a messagebox saying "You must enter a number in the range of 1 to 100 in
        the first box “
        Blank out the text in txtNumber1
        Set the cursor focus to txtNumber1
        Quit the subroutine
End If

Set  intNumber1 to hold the value contained in txtNumber1

Check the range entered by comparing intNumber1 with 100 and 1 –
if it is outside that range then
        Display a messagebox saying "You must enter a number in the range of 1 to 100 in
        the first box”
        Blank out the text in txtNumber1
        Set the cursor focus to txtNumber1
        Quit the subroutine
End If

Check that there is no decimal place contained in txtNumber1
if there is then
        Display a messagebox saying "You must enter an integer in the range of 1 to 100 in
                the first box”
        Blank out the text in txtNumber1
        Set the cursor focus to txtNumber1
        Quit the subroutine
End If

If the length of the entry in txtNumber2  = 0 Then
        Display a message box  saying "You must enter a number in the second box!"
        Set the cursor focus to txtNumber2
        Quit the subroutine
End If

If the length of the entry in txtNumber2 is > 3 Then
        Display a messagebox saying "You must enter a number in the range of 1 to 100 in
                the second box”
        Blank out the text in txtNumber2
        Set the cursor focus to txtNumber2
        Quit the subroutine
```

End If

If the entry in txtNumber2 isn't a number
    Display a messagebox saying "You must enter a number in the range of 1 to 100 in the second box"
    Blank out the text in txtNumber2
    Set the cursor focus to txtNumber2
    Quit the subroutine
End If

Set intNumber2 to hold the value contained in txtNumber2

Check the range entered by comparing intNumber2 with 100 and 1 –
if it is outside that range then
    Display a messagebox saying "You must enter a number in the range of 1 to 100 in the second box"
    Blank out the text in txtNumber2
    Set the cursor focus to txtNumber2
    Quit the subroutine
End If

Check that there is no decimal place contained in txtNumber2
if there is then
    Display a messagebox saying "You must enter an integer in the range of 1 to 100 in the second box"
    Blank out the text in txtNumber2
    Set the cursor focus to txtNumber2
    Quit the subroutine
End If

Set the height of the form to be 284
Bring the group box gbOperator to the front
Disable the group box gbNumbers

End of btnProcess_Click

Click event for File-> Exit

Dispose of the form
End the project

End of FileExit_Click

Click event for File-> New

Perform the Subroutine ClearStuff

End of FileNew_Click

Subroutine ClearStuff

Set the Text properties of txtNumber1 and txtNumber2 to equal “”
Set the Text property of lblAnswer to equal “”
Set the values in intNumber1, intNumber2 and decTotal to equal 0
Set the height of the form to equal 182
Enable the GroupBox gbNumbers
Enable the GroupBox gbOperator
Set the cursor focus to be on txtNumber1

End of ClearStuff

Click event for Help menu item

Store the current height of the form in the variable intSaveHeight
Set the height of the form to equal 389
Set the GroupBox gbAbout to be visible
Bring the GroupBox gbAbout to the front

End of Help_Click

---

btnReturn_Click

Set the height of the form to equal the value stored in intSaveHeight
Set the GroupBox gbAbout to invisible

End of btnReturn_Click

Click event for btnAdd, btnMinus, btnMultiply and btnDivide

Examine the Text property of the sender button (i.e. the button that was clicked by the user)
If it is “+” Then
    Add intNumber1 and intNumber2 and store the result in decAnswer
If it is “-” Then
    Subtract intNumber2 from intNumber1 and store the result in decAnswer
If it is “*” Then
    Multiply intNumber1 by intNumber2 and store the result in decAnswer
If it is “/” Then
    Examine the contents of intNumber2

If it is more than 0 Then
Divide intNumber1 by intNumber2 and store the result in decAnswer
Otherwise
Display an error message and exit the subroutine
End Ifs

Display the value in decAnswer in the text property of the label lblAnswer, along with continuation instructions
Set the height of the form to 389
Disable the GroupBox gbOperator

End of btnAdd_Click, btnMinus_Click, btnMultiply_Click and btnDivide_Click

You have now heard about design documentation and soon you will hear about robust testing techniques; both of these contribute towards what is known as 'technical documentation'. Technical documentation is aimed at fellow developers or members of the team who will be responsible for installation and maintenance of your software and as such can contain as much jargon as you feel necessary. However, you are also responsible for writing what is known as 'user documentation', which is aimed specifically at the user (although in many cases developers who are responsible for future project moderation will also read the user documentation that comes with software to get a better feel as to what the purpose of the software is). Because it is aimed at the user it must NOT contain jargon, and if it is totally unavoidable to include jargon then you should include a glossary of terms at the end.

User instructions (or user manuals to use a different name for the same thing) must be stand-alone documents, complete with cover, contents page and lots of illustrations to help you to explain to the user what the software does, why it does it and if it is necessary, how it does it. Your language should be clear and simple and under no circumstances should there be any hint of condescension or bad language (the author has been marking student user documentation for many years and has seen all manner of wordage brought into play in user manuals, some of which you would be embarrassed to repeat to your grandmother…). Remember that your users are your clients, and you would like them to continue supporting you in your work – this involves being polite to them!

Generally speaking, users require simple, easy to follow instructions that are easy to find when they are needed and aren't confusing or ambiguous. They want to be able to pick up the software and the manual and start working without the need for many hours of practice and / or training. Make sure that the most-often performed tasks in the software are documented thoroughly and that that documentation is easy to find. These instructions could be on-screen or paper-based but the same rules apply. Depending on your software you might have to provide a very thick, comprehensive manual or just simple-step-by-step instructions, or a combination of the two.

Tips for writing user instructions:

- Prefix each section of the instruction with a clear title stating what these instructions are going to explain.
- Don't 'hide' vital instructions on how to perform a frequently-needed task inside a section explaining something else.
- Start your instructions with a verb – a 'doing' word – like this list of tips, for example.
- Use a numbered list if it is vital that you perform certain tasks in that given order.
- Define any tasks that should have been done before you complete the task in hand. For example, if you want to wash your hands you will need to have found a water source before you can do that.
- Tell the user what to do if an error occurs. Preferably your user instructions should have a section all about what error messages they can expect to see and under what conditions.
- Put any warnings in a place where they are clearly visible and obvious.
- Tell the user what they ought to be doing next, when this particular task is completed.
- Write using language that is appropriate for your user.

Sometimes it is necessary to put the user instructions in context, by giving an overview of where this particular piece of software will fit within the business. If this helps your users to understand they whys and wherefores of your software, then it is a worthwhile task for you to do.

When you are writing your user instructions it is important to take into account your audience; remember that they might be used by more than one level of user. For example, your software might be used by people whose only job is to input information into your system – they want a quick-start, easy to read set of instructions that will enable them to start work straight away. It also might be used by the company accountant – he or she will need a much more detailed set of information as to how the information output from the system is calculated, and where it is likely to go after it has been processed. It might also be used by somebody who is manning a 'help desk' – users who are having issues with the software might ring the help desk for support so help desk operators are going to need a very in-depth analysis of the software in front of them. There might be a case for these users having access to the technical documentation as well as the user documentation, of course.

Given this, it might be a plan to divide your user instructions into sections that are aimed at different users – if the software warrants it, it might even mean that different user manuals are necessary, one for each type of user.

Key points to take into consideration when you are writing your instructions:

- If a picture is necessary then include one, but not at the expense of explaining in words what the user is to do.
- Keep your language clear, concise and unambiguous. If it is necessary to include a number, give the range of the numbers that are valid, for example.
- Don't use flowery and over-descriptive language.
- Make sure that your instructions are readily navigated – your users don't want to have to read through 10 pages of irrelevant waffle before they find the information they need.
- Put yourself in the shoes of your users when you re-read your user documentation. Let somebody else read your instructions before you publish so that you can get somebody else's opinion on your writing.
- Never EVER under ANY circumstances should you refer to Visual Basic .NET or programming or editors since this is likely to send the user screaming and running for the hills.
- Your user instructions should be a stand-alone document.

Following on in this section is an example user manual for the software detailed in the section 'Design documentation'. It does not have a front cover, contents page and separate page numbers, but the instructions are what counts here!

Introduction

Welcome to the TwoPlusTwo© software from SNC PLC! You have purchased this software specifically to help you with your business need when it comes to adding up two numbers and displaying a reliable and accurate total. You have the facility to add as many pairs of numbers as necessary and you are guaranteed an accurate answer, regardless of how many times you use your software.

Installation

The first time you insert the CD into your computer's optical media drive, the Installation Wizard will load and help you to install the software. Just follow the on-screen prompts by answering 'Yes' and by the time you have finished the process you will have a link to your new software both in your computer's Start menu and also as an icon on your main Windows opening screen, as shown here.

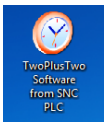
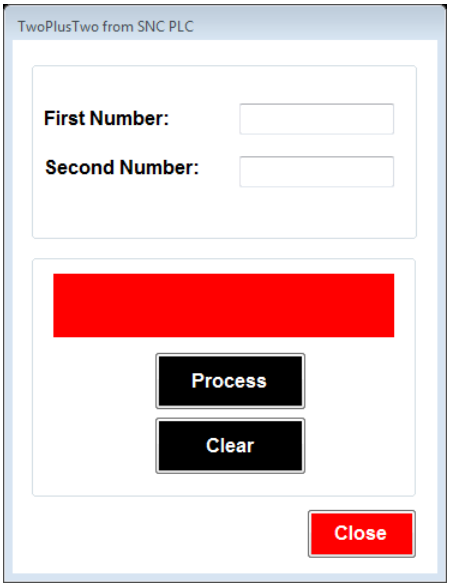


Figure 1 - TwoPlusTwo installed

Starting your software

To start your TwoPlusTwo© software, double-click your mouse in Figure 1, from your main Windows screen.

When your software starts, you will see the main screen as shown in Figure 2.



What do I do if my software doesn't start?

First of all make sure that it has been installed correctly, by following the procedure in the section 'Installation'. If not, install your software.

If this doesn't solve the problem, then you should contact the SNC PLC Technical Support line on telephone number 0115 9146414 (open 8:30am to 5pm, Monday to Friday).

Figure 2: TwoPlusTwo© main screen

Entering information into TwoPlusTwo

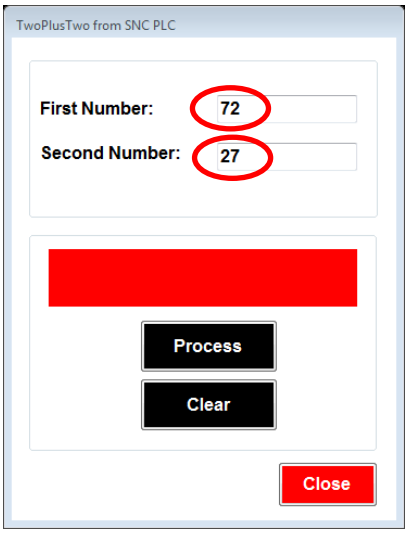
When your TwoPlusTwo© starts, your cursor will be placed next to the prompt "First Number". Here, you need to enter type your first of the two numbers that you want to total.

What do I enter: the number you enter must be a whole number, and in the range of 1 to 9999.

Next you should press either the Tab key on your keyboard, or use your mouse to click on the box alongside the "Second Number" prompt.

What do I enter: the number you enter must be a whole number, and in the range of 1 to 9999.

Now look at Figure 3. You will see here that the numbers 72 and 27 have been entered into TwoPlusTwo©.



What do I do if I enter the wrong number?

Don't panic! You can either put the cursor to the right of the incorrectly typed number and press the Backspace key on your keyboard, or you can click the black button on the form called 'Clear'. More on this later.

Figure 3: Numbers entered in TwoPlusTwo

Calculating the total

Once you have entered both numbers, then you should click the black button marked 'Process'.

You will see the total of your two numbers displayed in the red box beneath your entry. For your convenience the numbers you entered are left displayed, as shown in Figure 4.

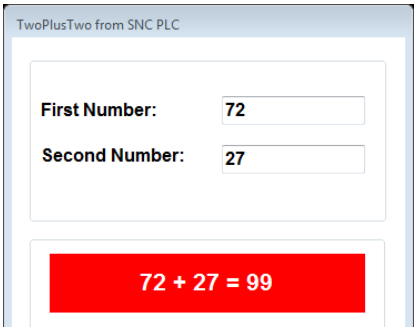


Figure 4: TwoPlusTwo© showing your answer

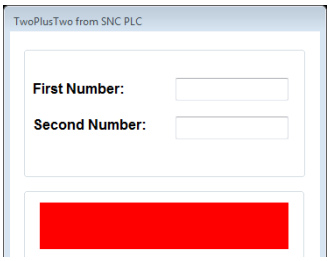
What do I do if the answer is incorrect?

In the unlikely event of an error occurring, you should contact the SNC PLC Technical Support line on telephone number 0115 9146414 (open 8:30am to 5pm, Monday to Friday).

Clearing an old set of numbers ready to enter another set

When you have finished with the numbers that you have just entered, you need to clear them to enable you to enter a new set.

Click on the black button marked 'Clear'.



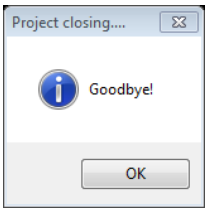
The numbers you entered have now gone, and the label has cleared. Your cursor is now back in the box alongside the prompt 'First Number' ready for you to enter a new set of numbers, as shown in Figure 5.

Figure 5: Cleared information, ready for new input

Closing TwoPlusTwo©

When you are ready to close your software, you should click once with your mouse on the red button marked 'Close'.

When you've clicked once on the Close button, TwoPlusTwo© will bid you 'Goodbye'.



Click on the button marked 'OK' and the software will be completely closed.

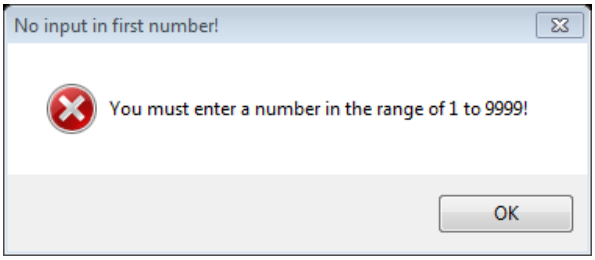
TUTORIAL

We are going to work through three sums. You will be asked to enter some information into TwoPlusTwo© and some of it will look incorrect – however, it is important that you have some experience of mistakes so that when you are using the software properly you will have seen the error messages that TwoPlusTwo© will give you, and you will know what to do and how to put the problem right.

Lesson 1 – Entering information into 'First Number'

Nothing entered into "First Number" at all

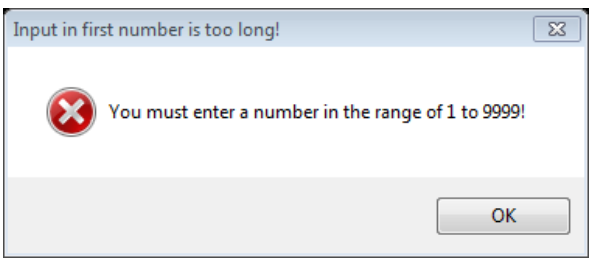
Load TwoPlusTwo© and at the prompt 'First Number', **don't enter anything** before you click the 'Process' button. TwoPlusTwo© will generate an error message :



Look at the top line of this error message: it says 'No input in first number!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

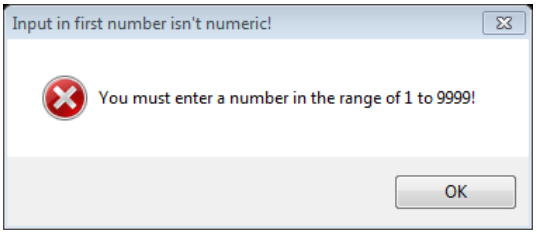
You entered something that is too long

Now enter '**11111111**' into the box alongside the prompt 'First Number' and click the 'Process' button.



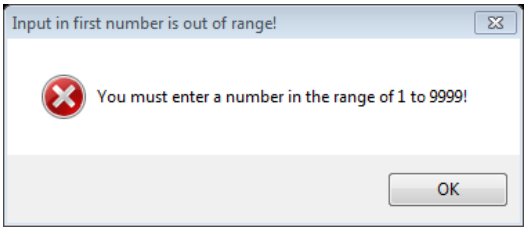
You will be shown an error message. Look at the top line of this error message: it says 'Input in first number is too long!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

You entered something into "First Number" that is not numeric
Type your own first name into the box alongside the prompt 'First Number' and click the 'Process' button.



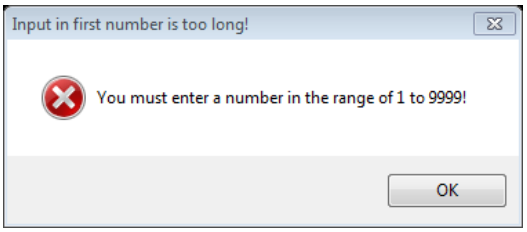
You will be shown an error message. Look at the top line of this error message: it says 'Input in first number isn't numeric!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

You enter a number that is out of the accepted range of 1 to 9,999
Type **0** into the box alongside the prompt 'First Number' and click the 'Process' button.



You will be shown an error message. Look at the top line of this error message: it says 'Input in first number out of range!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

Now enter **10000** into the box alongside the prompt 'First Number' and click the 'Process' button.

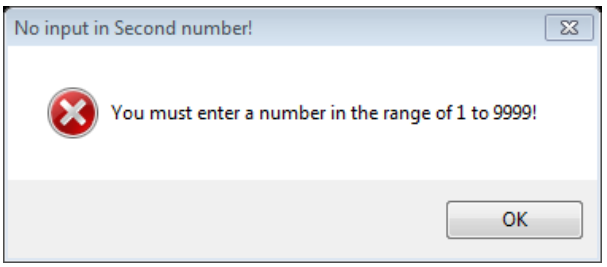


You will be shown an error message. Look at the top line of this error message: it says 'Input in first number is too long!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

To finish off Lesson 1, enter **75** into the box alongside 'First Number' and click on the 'Process' button.

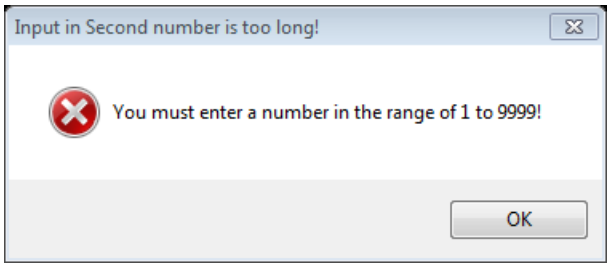
Lesson 2 – Entering information into 'Second Number'

Nothing entered into "Second Number" at all
Load TwoPlusTwo© and at the prompt 'Second Number', **don't enter anything** before you click the 'Process' button. TwoPlusTwo© will generate an error message :



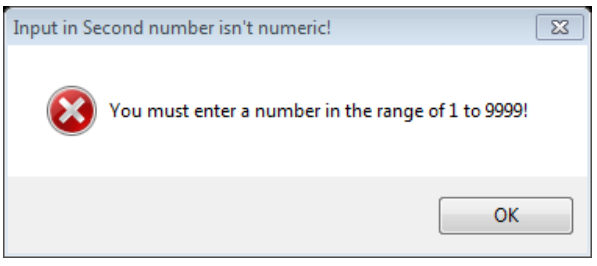
Look at the top line of this error message: it says 'No input in Second number!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

You entered something that is too long
Now enter '**11111111**' into the box alongside the prompt 'Second Number' and click the 'Process' button.



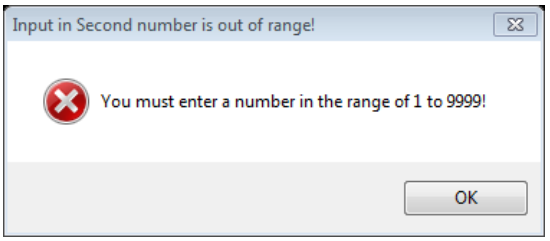
You will be shown an error message. Look at the top line of this error message: it says 'Input in Second number is too long!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

You entered something into "Second Number" that is not numeric
Type your own Second name into the box alongside the prompt 'Second Number' and click the 'Process' button.



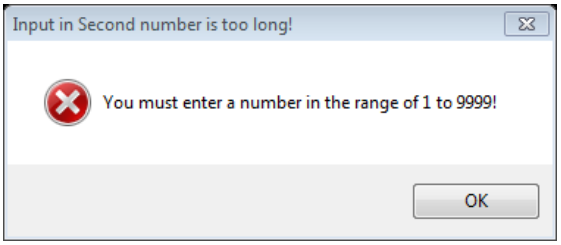
You will be shown an error message. Look at the top line of this error message: it says 'Input in Second number isn't numeric!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

You enter a number that is out of the accepted range of 1 to 9,999
Type **0** into the box alongside the prompt 'Second Number' and click the 'Process' button.



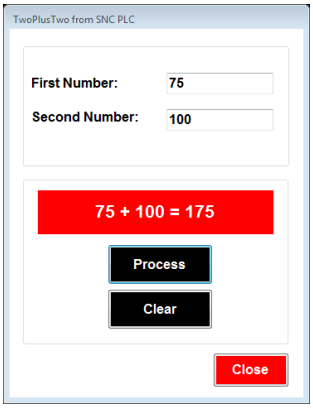
You will be shown an error message. Look at the top line of this error message: it says 'Input in Second number out of range!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

Now enter **10000** into the box alongside the prompt 'Second Number' and click the 'Process' button.



You will be shown an error message. Look at the top line of this error message: it says 'Input in Second number is too long!' in the top line of this message, and 'You must enter a number in the range of 1 to 9999!' as the main message. Click the button that says 'OK' and you will be back to the main screen of TwoPlusTwo©.

Now enter **100** into the box alongside the prompt 'Second Number' and click the 'Process' button.



We now have two valid numbers - 75 and 100 - and TwoPlusTwo© shows us the answer.

You are now ready to use your TwoPlusTwo© software!

Once you have started to implement coding behind your projects you will have to learn how to perform robust testing techniques.

There are several aspects behind testing, with this being the basic aspect:

- Does each form in the project load correctly, with all objects in the right place and with all spelling on the form correct?
- Does only one instance of a form load?
- Is the tab order set correctly? (This means that when you press the tab key control passes to each object in the right order.)
- Does each object allow for enough input to be entered and / or displayed?
- Does each form close correctly?

Once you've established the basic aspect of the functionality of the project, then the data being input to the project must be tested. The tests that should be made here are:

- That data has actually been entered. (Length = 0)
- That not too much data has been entered. (Length > …)
- That data is of the appropriate type (numeric, string etc).
- That numeric data is within the valid range.

This list is by no means exhaustive, but is more the minimum testing that you should perform.

After you've tested that input is valid, you must check that it has been processed correctly by the project. This is most usually done by examining the output from the project; output can be displayed within a form, printed in a report, stored within a file or database, or a combination of each of these.

Robust testing must be planned in a considered and structured way. The method you will use in this unit is to draw up a table, with each input having a column within the table detailing the input you will make.

Designing structured test data

1. In the table below, the column headed 'Test Num' is purely the sequential number of the test and will start at 1 and end once you've covered every test.
2. The next columns should contain details of the input into each of the objects on the form.
3. The next column should be the group number or name for this test (or these tests). The basic aspect of testing (see above) would cover all the 'does it load'-type testing; "Group 1" might be where you enter numbers into two text boxes (for example entering 5 into txtFirstNum and 6 into txtSecondNum – does lblAnswer display 11?) "Group 2" might be where you enter 15 and 25 – is the answer 40? And so on.
4. The final column should explain why you are doing that test.

Over the page is a sample plan of structured test data for Exercise 2.1, from page 117. It should address each of the aspects listed above in terms of basic functionality and also data-related.

Structured Test Data for Exercise 2.1			
Test Num	**Form1**	**Group**	**Why**
1	Load	Basic	To check that the form loads correctly and that all objects are in the right place with the right spelling and with the appropriate colours.
2	Focus	Basic	To check that when the form loads the cursor is placed in txtForename
3	TabOrder	Basic	To check that when the Tab key is pressed the cursor moves to txtSurname

Test Num	Input into txtForename	Input into txtSurname	Group	Why
4	""		Basic	To see if there is an error message displayed if the user doesn't enter anything into txtForename.
5	Jane	""	Basic	To see if there is an error message displayed if the user doesn't enter anything into txtSurname.
6	111111111111 111111111111		Basic	To see if there is an error message displayed if the user enters too many characters (over 20) into txtForename.
7	Jane	111111111111 111111111111	Basic	To see if there is an error message displayed if the user enters too many characters (over 20) into txtSurname.
8	Jane	Fletcher	1	To see that lblMessage displays "Jane Fletcher, welcome to programming!"
9	Peter	Rabbit	2	To see that lblMessage displays "Peter Rabbit, welcome to programming!"

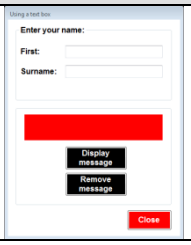
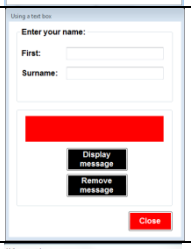
Test Num	btnRemove	btnDisplay	btnClose	Group	Why
10		Click		3	To check that the message is displayed in lblMessage with the appropriate name.
11	Click			4	To check that the message is removed from lblMessage, txtForename and txtSurname are cleared and focus is put in txtForename.
12			Click	5	To check that the project closes.

Notice that there are 12 tests planned for this very simple program. This is NOT overkill and should point you in the direction of how many tests you should be performing when you are testing your programs!

Structured test plan

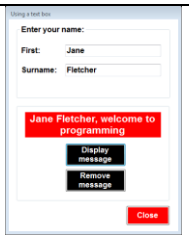
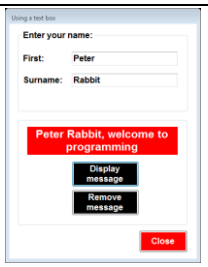
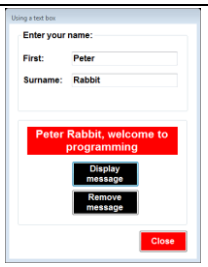
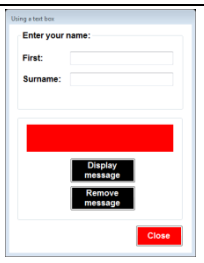
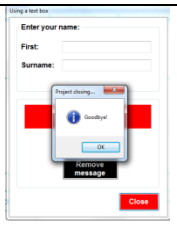
Designing the test data to be put into your program is just the first part of the testing procedure. The next stage is to take that test data and to implement it within a plan. Just having the plan isn't enough, however – you have to prove that you have implemented it!

Demonstrating proof can be done in one of two ways: the first way is to put the proof within the structured test plan itself, in the column headed 'Actual result' in the form of screen shots (see over the page); the second way is to include the screen shots in a numbered list after the test plan.

FORM NAME : Form1					
Test number	**Component(s) tested**	**Input**	**Expected result**	**Actual result**	**Yes/No**
1	Form1_Load	None	Form displays correctly		Y
2	Focus	None	Cursor is placed into txtForename on Form_Load		Y
3	TabOrder	Press Tab key	Cursor is placed in txtSurname after Tab key pressed		Y
4	btnDisplay_Click (txtForename)	No input into txtForename, click on btnDisplay	Error message to user stating that there has been no input into txtForename		Y
5	btnDisplay_Click (txtForename and txtSurname)	Jane and “”	Error message to user stating that there has been no input into txtSurname		Y
6	btnDisplay_Click and txtForename	1111111111111 1111111111111	Error message to the user stating that the entry for txtForename is too long		Y

Testing Projects

Lecturer: Jane Fletcher

Test number	Component(s) tested	Input	Expected result	Actual result	Yes/No
7	btnDisplay_Click (txtForename and txtSurname)	Jane and 1111111111111 1111111111111	Error message to the user stating that the entry for txtSurname is too long		Y
8	btnDisplay_Click (txtForename and txtSurname)	Jane and Fletcher	lblMessage displays “Jane Fletcher, welcome to programming”		Y
9	btnDisplay_Click (txtForename and txtSurname)	Peter and Rabbit	lblMessage displays “Peter Rabbit, welcome to programming”		Y
10	btnDisplay_Click (txtForename and txtSurname)	Peter and Rabbit	lblMessage displays “Peter Rabbit, welcome to programming”		Y
11	btnRemove	Click	Text boxes clear, label clears and cursor is placed in txtForename		Y
12	btnClose	Click	Program closes		Y

If the test plan should reveal that there are bugs within the project, then you should document those – the final column of the plan indicates if the test was successful ('yes' or 'no') so in the case of the test failing, you should enter 'N' in this column and go on to explain in a separate document why the test failed, and explain what you did to solve the problem. You should then repeat the test and any further tests that were affected by the initial failure and document that.

Below is test number 8 repeated, but in this case there is an error.

Test number	Component(s) tested	Input	Expected result	Actual result	Yes/No
8	btnDisplay_Click (txtForename and txtSurname)	Jane and Fletcher	lblMessage displays "Jane Fletcher, welcome to programming"		N See note 1

Note 1
Error occurred during test number 8 – instead of the program displaying "Jane Fletcher, welcome to programming" it displays "JaneFletcher, welcome to programming".

The coding before correction is:

```
'Concatenate the content of the text box with a message and display it to the user.
lblMessage.Text = txtForename.Text & txtSurname.Text & ", welcome to programming"
```

The correction required is to alter the concatenation to include an extra space, as shown below:

```
'Concatenate the content of the text box with a message and display it to the user.
lblMessage.Text = txtForename.Text & " " & txtSurname.Text & ", welcome to programming"
```

The repeated test now indicates that this error has been removed.

Test number	Component(s) tested	Input	Expected result	Actual result	Yes/No
8	btnDisplay_Click (txtForename and txtSurname)	Jane and Fletcher	lblMessage displays "Jane Fletcher, welcome to programming"		Y

Example programs

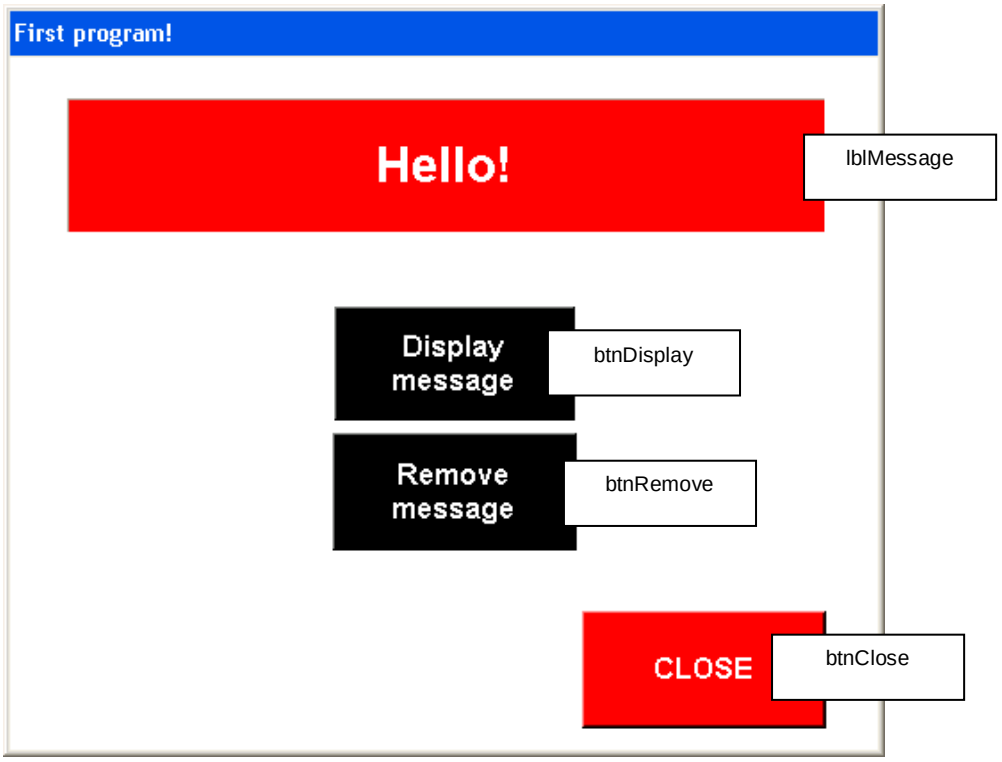
Lecturer: Jane Fletcher

This section will provide you with the specifications for and coding of the programs that will be developed in class. It will also provide you with the specifications for some extension programs that you will be required to write each week, based on the ones developed in class.

It is important that you write each program given to you in the order that they're set since they have been designed with a 'building block' idea in mind – if you miss out a brick in your learning wall, then you aren't building a very stable foundation and your wall stands an excellent chance of falling down and crushing your feet. Or something similar – whatever the programming equivalent of a pair of crushed feet is. It doesn't matter how many times you attempt to write a program – when the author was learning to program sometimes she would have a go at the same program several times, until she'd learned what she was doing and could repeat it without having to rely on copying or notes – as long as when you've finished and finally move on, you understand what you did and why you did it.

Sometimes students complain that they don't understand programming – usually this will happen after maybe the third or fourth lesson - and they start looking at the lecturer with dislike and dreading the sessions spent trapped in a classroom with her. The lecturer is aware that this happens and expects it. At an appropriate point she will launch into her usual speech: "Imagine that you were learning to speak a foreign language, like Klingon for example. After four and a half hours (three weeks' worth of lessons) of learning Klingon, how fluent do you think you'd be? You may be able to ask for the bathroom, or to ask directions to the nearest hospital in Klingon but that's probably about the extent of it." Well, programming is a foreign language to people new to it (the author is more fluent in Visual Basic .NET than she is in English) and so are the concepts of logic and simplicity. Programming statements are extremely simple once you understand them, but when you start out as a fledgling developer you tend to imagine that you should be thinking in a far more complex way than you really need to. If you can write a simple step-by-step list, with each step being very small and not complex, than you can program. Trust me; I'm a programmer. If I can do it, anybody can.

1.0 Specification for program developed in class



A program is needed to meet the following requirements:

Upon first load, lblMessage should not be visible. Make sure that you set the Visible property of lblMessage to be False in the Properties Window.

When the user clicks btnDisplay, lblMessage should be displayed by altering the Visible property of lblMessage to be True in the Click event of btnDisplay.

When the user clicks btnRemove, lblMessage should be taken off the screen by altering the Visible property of lblMessage to be False in the Click event of btnRemove.

When the user clicks btnClose, the program should close. Use the key word 'End' in the click event of btnClose.

Don't forget to comment your program.

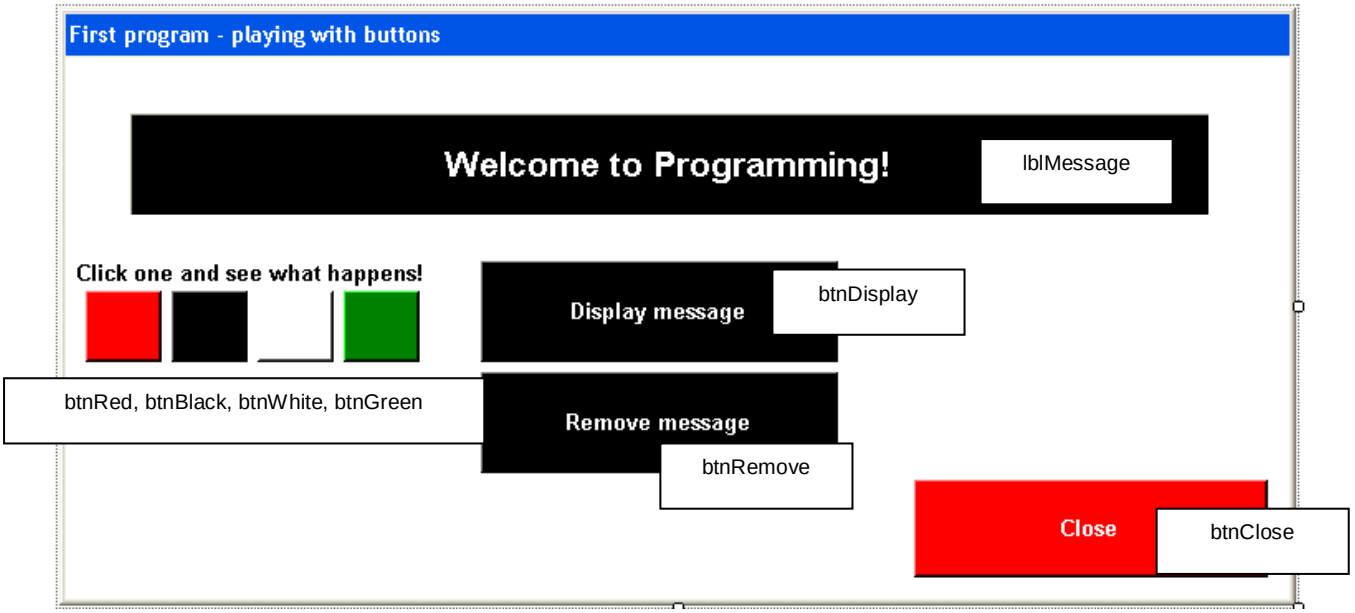
Now - experiment!

Try making the label a different colour.
Try altering the font of the objects.
Add other messages – remember though, be polite….

Coding for the program

```
Public Class Form1

    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClose.Click

        'Stop the project running.
        End
    End Sub

    Private Sub btnDisplay_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnDisplay.Click

        'Display the label by altering the Visible property.
        lblMessage.Visible = True
    End Sub

    Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnRemove.Click

        'Remove the label from the screen by altering the Visible property.
        lblMessage.Visible = False
    End Sub

End Class
```

Exercise 1.1



A program is needed to meet the following requirements:

Upon first load, lblMessage should not be visible.
When the user clicks btnDisplay, lblMessage should be displayed.
When the user clicks btnRemove, lblMessage should be taken off the screen.

When the user clicks btnRed, the background of lblMessage should become red, and the text colour should be white. This requires you to access the ForeColor and BackColor properties of the label, on the click event of btnRed.

When the user clicks btnBlack, the background of lblMessage should become black, and the text colour should be white. This requires you to access the ForeColor and BackColor properties of the label, on the click event of btnBlack.

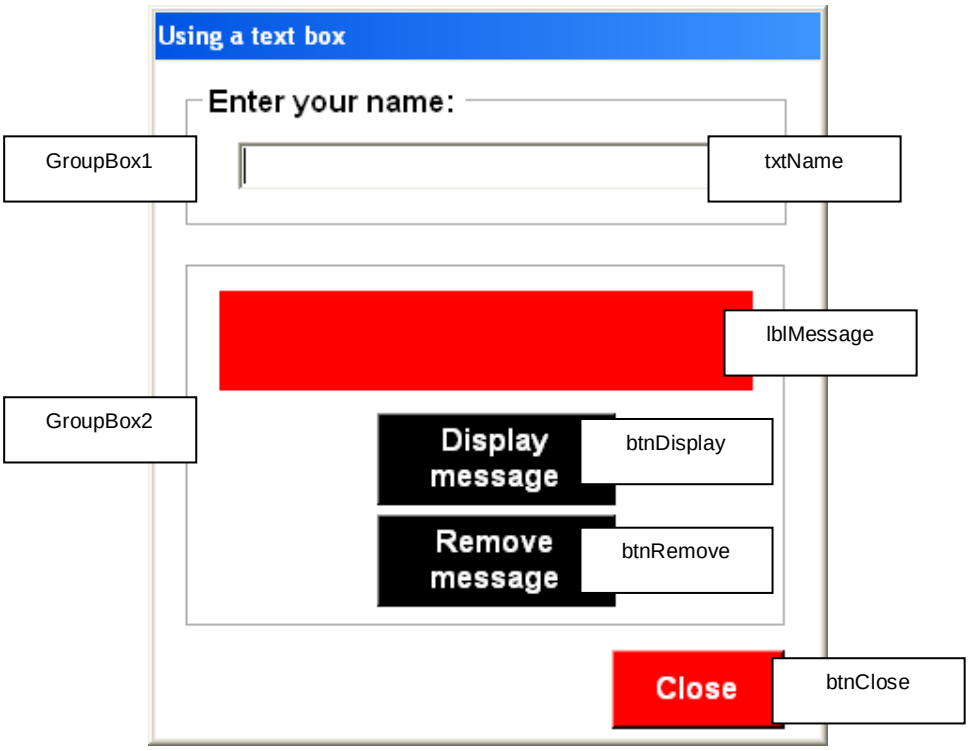
When the user clicks btnWhite, the background of lblMessage should become white, and the text colour should be black. This requires you to access the ForeColor and BackColor properties of the label, on the click event of btnWhite.

When the user clicks btnGreen, the background of lblMessage should become green, and the text colour should be white. This requires you to access the ForeColor and BackColor properties of the label, on the click event of btnGreen.

When the user clicks btnClose, the program should close.

You should make sure that your program is commented, and save it in a brand new folder within your 'Week 01' folder.

2.0 Specification for program developed in class



A program is needed to meet the following requirements:

Upon first load, all objects on the form should be visible and the cursor should be flashing in txtName.
When the user clicks btnDisplay, the name entered into txtName should be displayed, along with the message ", welcome to programming" in lblMessage.
When the user clicks btnRemove, lblMessage should blanked out, the text in txtName should be removed, and the cursor should be put into txtName.

When the user clicks btnClose, the program should close.

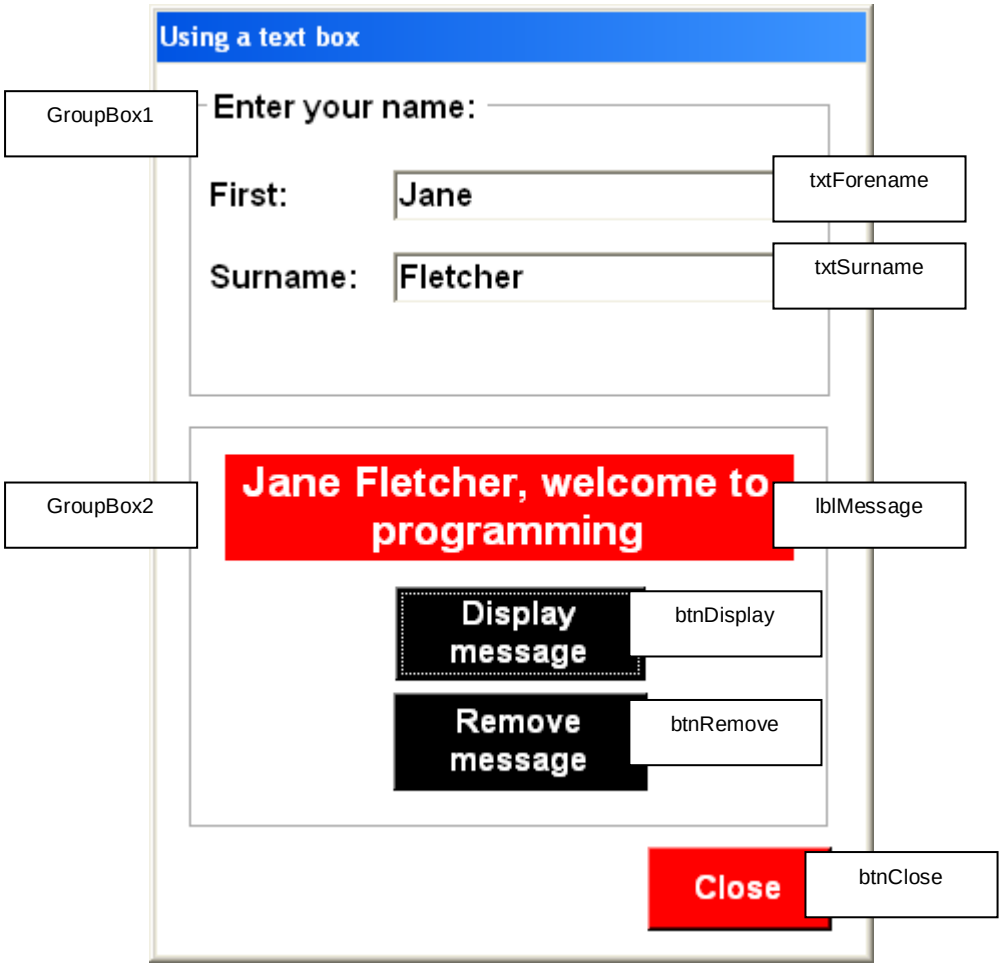
Now - experiment!

Put two text boxes on the form – forename and surname – and concatenate them (with a space in between, please) and display a longer message.

Coding for the program

```
Public Class Form1

    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClose.Click

        'Close the project down
        End
    End Sub

    Private Sub btnDisplay_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnDisplay.Click

        'Concatenate the content of the text box with a message and display it to the user.
        lblMessage.Text = txtName.Text & ", welcome to programming"
    End Sub

    Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnRemove.Click

        'Clear down the message from the label, remove the text from the text box and put
        'the focus of the program back into the text box.
        lblMessage.Text = ""
        txtName.Text = ""
        txtName.Focus()
    End Sub
End Class
```

Exercise 2.1



A program is needed to meet the following requirements:

Upon first load, all objects on the form should be visible and the cursor should be flashing in txtName.

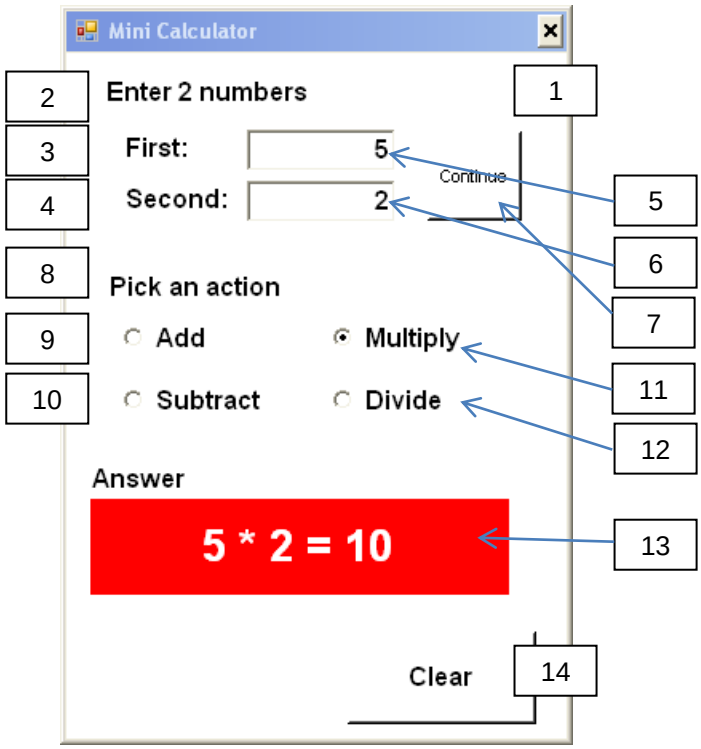
When the user clicks btnDisplay, the name entered into txtForename and txtSurname should be displayed, along with the message ", welcome to programming" in lblMessage.

When the user clicks btnRemove, lblMessage should blanked out, the text in txtForename and txtSurname should be removed, and the cursor should be put into txtForename.

The cursor should tab between Forename and Surname accurately.

When the user clicks btnClose, the program should close.

3.0 Specification for program developed in class



Item	Object
1	Form1
2	GroupBox1
3	Label1
4	Label2
5	TextBox1 (txtFirstNumber)
6	TextBox2 (txtSecondNumber)
7	Button1 (btnContinue)
8	GroupBox2
9	RadioButton1 (radAdd)
10	RadioButton2 (radSubtract
11	RadioButton3 (radMultiply
12	RadioButton4
13	Label3 (lblAnswer)
14	Button2 (btnClear

Example programs – 3. Introducing RadioButtons

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

When the program loads, the height of the form should be set so that only the first group box (GroupBox1) is visible.

Once the user has entered numbers into both text boxes and clicked btnContinue, the form's height should increase to show GroupBox2.

Once the user has selected one of the radio buttons, the form's height should increase to show everything, including btnClear.

At the top of the program (after the Public Class) declare three variables like so:

```
Dim intFirstNumber, intSecondNumber As Integer
Dim decTotal As Decimal
```

The coding should assign the value that is contained in txtFirstNumber into intFirstNumber like so:

```
intFirstNumber = txtFirstNumber.Text
```

and then perform a similar operation for the value in txtSecondNumber.

If the user clicks radAdd, then the program must add the two numbers together (in the default CheckChanged event for the radio button) and display the total in lblAnswer in the format shown here:

First number entered + second number entered = the answer

Where *first number entered* is the value in intFirstNumber, and so on.

The other radio buttons should work in a similar way, but should perform the appropriate arithmetic operation and display.

btnClear, when clicked, should put the form back to its original size, clear both text boxes and put the cursor into txtFirstNumber.

Example programs - 3. Introducing RadioButtons

Lecturer: Jane Fletcher

Coding for the program

```
Public Class Form1

    Dim intFirstNumber, intSecondNumber As Integer
    Dim decTotal As Decimal

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        'Shrink the size of the form.
        Me.Height = 191
    End Sub

    Private Sub btnContinue_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnContinue.Click

        'Move the values from the text boxes into the
        'integer variables, but only if they are valid.
        'Non valid input should be thrown out.

        If txtFirstNumber.Text.Length = 0 Then
            MsgBox("You have not entered anything into the first text box!", _
                     MsgBoxStyle.Critical, "ERROR!")
            txtFirstNumber.Focus()
            Exit Sub
        End If

        If txtFirstNumber.Text.Length > 4 Then
            MsgBox("You have entered a value that is too long!", MsgBoxStyle.Critical, "Error!")
            txtFirstNumber.Text = ""
            txtFirstNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtFirstNumber.Text) Then
            MsgBox("You have entered a non-numeric value!", MsgBoxStyle.Critical, "Error!")
            txtFirstNumber.Text = ""
            txtFirstNumber.Focus()
            Exit Sub
        End If

        intFirstNumber = txtFirstNumber.Text

        If (intFirstNumber < 1) Or (intFirstNumber > 1000) Then
            MsgBox("You have entered a number out of range!", MsgBoxStyle.Critical, "ERROR!")
            txtFirstNumber.Text = ""
            txtFirstNumber.Focus()
            Exit Sub
        End If

        If txtFirstNumber.Text.Length = 0 Then
            MsgBox("You have not entered anything into the first text box!", _
                     MsgBoxStyle.Critical, "ERROR!")
            txtFirstNumber.Focus()
            Exit Sub
        End If

        If txtSecondNumber.Text.Length > 4 Then
```

Example programs - 3. Introducing RadioButtons

Lecturer: Jane Fletcher

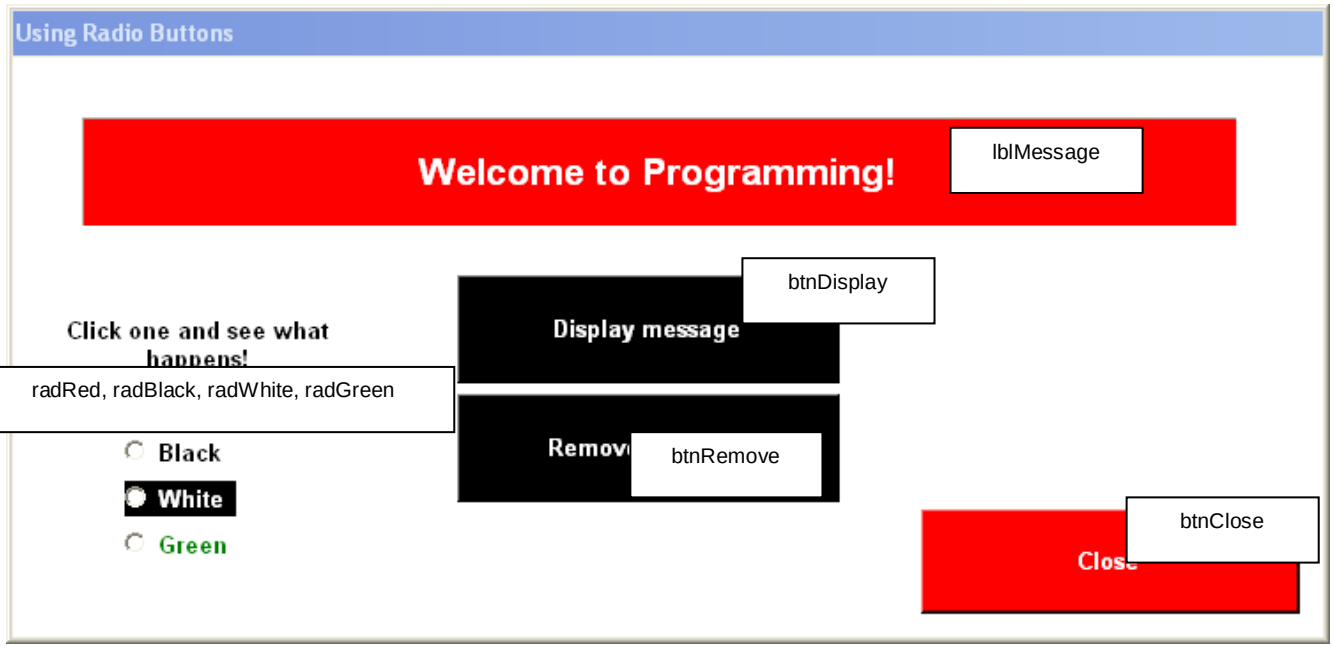
```
            MsgBox("You have entered a value that is too long!", MsgBoxStyle.Critical, "Error!")
            txtSecondNumber.Text = ""
            txtSecondNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtSecondNumber.Text) Then
            MsgBox("You have entered a non-numeric value!", MsgBoxStyle.Critical, "Error!")
            txtSecondNumber.Text = ""
            txtSecondNumber.Focus()
            Exit Sub
        End If

        intSecondNumber = txtSecondNumber.Text

        If (intSecondNumber < 1) Or (intSecondNumber > 1000) Then
            MsgBox("You have entered a number out of range!", MsgBoxStyle.Critical, "ERROR!")
            txtSecondNumber.Text = ""
            txtSecondNumber.Focus()
            Exit Sub
        End If

        'Alter the height of the form.
        Me.Height = 364
    End Sub

    Private Sub radAdd_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles radAdd.CheckedChanged

        'The user wants to add up the two numbers.
        decTotal = intFirstNumber + intSecondNumber
        lblAnswer.Text = intFirstNumber & " + " & _
            intSecondNumber & " = " & decTotal

        'Increase the size of the form
        Me.Height = 546
    End Sub

    Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
             btnClear.Click

        'Shrink the form and clear the text boxes.
        Me.Height = 191
        txtFirstNumber.Text = ""
        txtSecondNumber.Text = ""
        txtFirstNumber.Focus()
    End Sub

    Private Sub radSubtract_CheckedChanged(ByVal sender As System.Object, ByVal e As _
             System.EventArgs) Handles radSubtract.CheckedChanged

        'The user wants to subtract the two numbers.
        decTotal = intFirstNumber - intSecondNumber
        lblAnswer.Text = intFirstNumber & " - " & _
            intSecondNumber & " = " & decTotal

        'Increase the size of the form
        Me.Height = 546
    End Sub
```

Example programs - 3. Introducing RadioButtons

Lecturer: Jane Fletcher

```
    Private Sub radMultiply_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles radMultiply.CheckedChanged

        'The user wants to multiply the two numbers.
        decTotal = intFirstNumber + intSecondNumber
        lblAnswer.Text = intFirstNumber & " * " & _
            intSecondNumber & " = " & decTotal

        'Increase the size of the form
        Me.Height = 546
    End Sub

    Private Sub radDivide_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles radDivide.CheckedChanged

        'The user wants to divide the two numbers.
        decTotal = intFirstNumber / intSecondNumber
        lblAnswer.Text = intFirstNumber & " / " & _
            intSecondNumber & " = " & decTotal

        'Increase the size of the form
        Me.Height = 546
    End Sub
End Class
```

Exercise 3.1



A program is needed to meet the following requirements:

Upon first load, lblMessage should not be visible.
When the user clicks btnDisplay, lblMessage should be displayed.
When the user clicks btnRemove, lblMessage should be taken off the screen.

When the user clicks radRed, the background of lblMessage should become red, and the text colour should be white.

When the user clicks radBlack, the background of lblMessage should become black, and the text colour should be white.

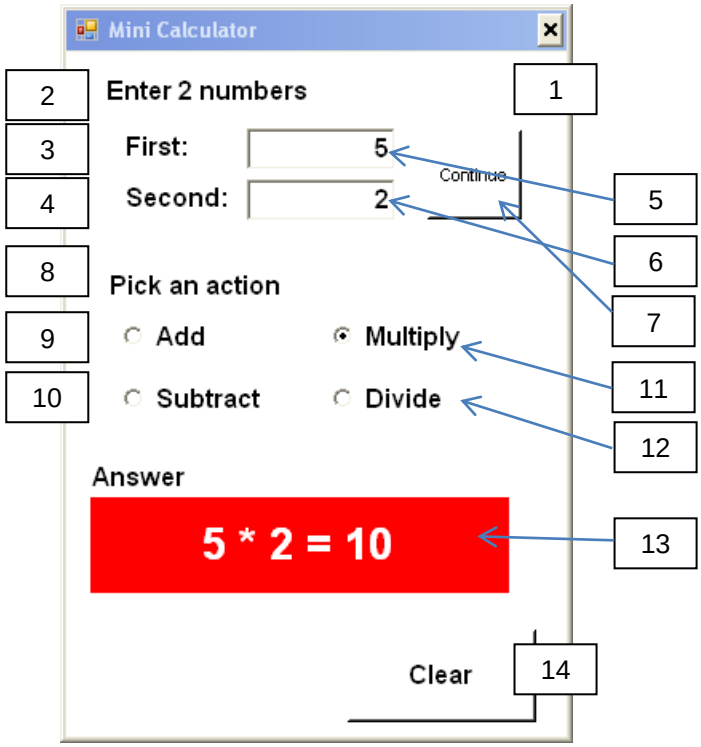
When the user clicks radWhite, the background of lblMessage should become white, and the text colour should be black.

When the user clicks radGreen, the background of lblMessage should become green, and the text colour should be white.

When the user clicks btnClose, the program should close.

Complete design documentation is required for this program.

Exercise 3.2



Item	Object
1	Form1
2	GroupBox1
3	Label1
4	Label2
5	TextBox1 (txtFirstNumber)
6	TextBox2 (txtSecondNumber)
7	Button1 (btnContinue)
8	GroupBox2
9	RadioButton1 (radAdd)
10	RadioButton2 (radSubtract
11	RadioButton3 (radMultiply
12	RadioButton4
13	Label3 (lblAnswer)
14	Button2 (btnClear

Example programs – 3. Introducing RadioButtons

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

Taking the program written originally in class, try entering letters into the text boxes and clicking btnProcess. What happens?

When entering any numeric value into a program, it must always be validated. The validation necessary is as follows:

- The value must be entered (i.e. there is something there!);
- The value must not be too long (i.e. causing an overflow);
- The value must be numeric;
- The value must be in the range specified. In this instance, the numbers allowed are in the range of 1 to 100. Anything outside that range is invalid.

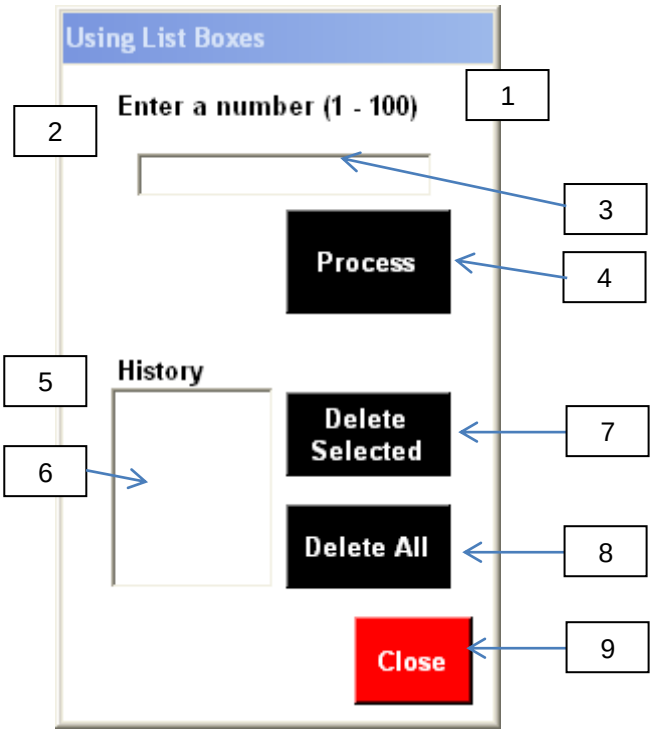
Write that validation!

Tips
Ask Google what "IsNumeric" does in Visual Basic .NET.
Play with the Text.Length property of a text box.
"Greater than" is written as >.
"Less than" is written as <.

What happens if you input 0 in txtSecondNumber and then click the "Divide" radio button? Whoops!

Remember to internally document your code.

4.0 Specification for program developed in class



Item	Object
1	Form1
2	GroupBox1
3	TextBox1 (txtNumber)
4	Button1 (btnProcess)
5	GroupBox2
6	ListBox1 (lstHistory)
7	Button2 (btnDeleteSelected)
8	Button3 (btnDeleteAll)
9	Button4 (btnClose)

A program is needed to meet the following requirements:

When the program loads, the entire form should be visible and the cursor placed in txtNumber.

When the user clicks btnProcess, the program should validate the entry in txtNumber as being numeric and within the range of 1 to 100. Any illegal input should generate a message box displaying a suitable message, the text within the text box should be deleted and the cursor placed into the text box ready for user input.

Once the number has been validated, it should be added to lstHistory. The text box must be cleared and the cursor must be placed in the text box ready for user entry.

When the user clicks btnDeleteSelected, the program must make sure that the user has actually selected something in lstHistory. If they have, then that entry should be deleted.

When the user clicks btnDeleteAll, the program must ask the user if they are sure that they wish to delete the entire contents of the list box via a message box. If the user answers yes, then the contents of the list box must be deleted, the contents of the text box deleted and the cursor placed in the box ready for user entry.

When the user clicks btnClose, the entire program should end.

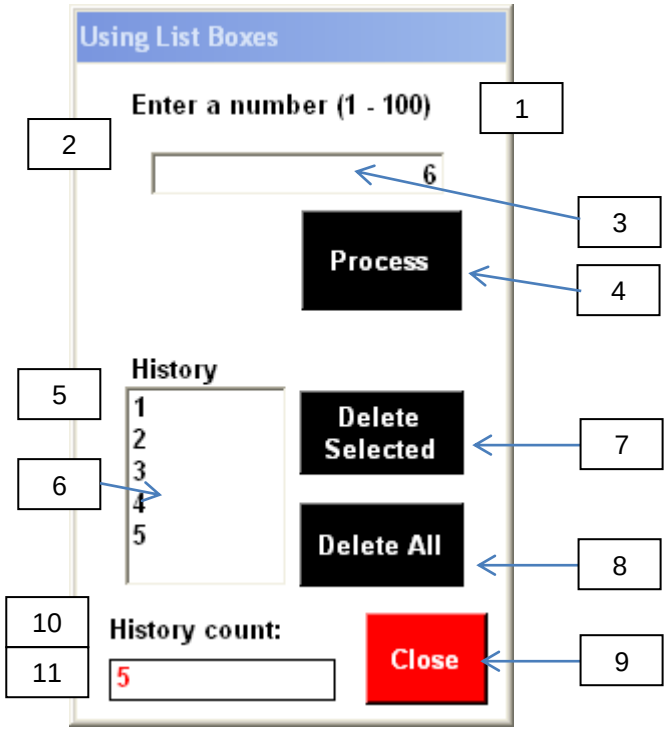
The program should use an integer variable called intNumber.

Coding for the program

```
Public Class Form1

    Dim intNumber As Integer

    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClose.Click

        'Close the project
        End
    End Sub

    Private Sub btnProcess_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnProcess.Click

        'Get the input from the user and validate it. The input
        'must be numeric and in the range of 1 to 100.

        If txtNumber.Text.Length = 0 Then
            'the user hasn't entered anything
            MsgBox("You haven't entered anything!", MsgBoxStyle.Critical, "Error!")
            txtNumber.Focus()
            Exit Sub
        End If

        If txtNumber.Text.Length > 3 Then
            'the user has entered too much
            MsgBox("Your entry is too long!", MsgBoxStyle.Critical, "Error!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtNumber.Text) Then
            'the entry isn't a number
            MsgBox("You haven't entered a number!", MsgBoxStyle.Critical, "Error!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        'The user has entered a number so store it in an integer variable.
        intNumber = txtNumber.Text

        If (intNumber < 1) Or (intNumber > 100) Then
            'The user's entry is numeric, but outside the valid range
            MsgBox("You have entered a number outside the range of 1 to 100!", _
                     MsgBoxStyle.Critical, "Error!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        'By the time we get to here the entry is valid so we can add it to the history list box.
        lstHistory.Items.Add(intNumber)
```

```
        'Clear the text box and put the focus up there for the next input.
        txtNumber.Text = ""
        txtNumber.Focus()
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
            Handles MyBase.Load

        'Put the focus into the text box
        txtNumber.Focus()
    End Sub

    Private Sub btnDeleteSelected_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnDeleteSelected.Click

        'The user has chosen to delete the selected item in the list box.

        'First, check that the user has selected something.
        If lstHistory.SelectedIndex = -1 Then
            'They haven't so throw them out.
            MsgBox("You must pick something to delete first!", MsgBoxStyle.Critical, "Error!")
            Exit Sub
        End If

        'If we get here then the user has picked something to delete.
        lstHistory.Items.RemoveAt(lstHistory.SelectedIndex)
        txtNumber.Focus()
    End Sub

    Private Sub btnDeleteAll_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
            Handles btnDeleteAll.Click

        'The user has selected to delete everything.

        'Warn them first!
        Dim intResponse As Integer
        intResponse = MsgBox("Do you really wish to delete everything?", MsgBoxStyle.YesNo, _
              "Are you sure?")
        If intResponse = vbYes Then 'They want to delete
            lstHistory.Items.Clear()
        End If
        txtNumber.Focus()
    End Sub

End Class
```

Exercise 4.1



Item	Object
1	Form1
2	GroupBox1
3	TextBox1 (txtNumber)
4	Button1 (btnProcess)
5	GroupBox2
6	ListBox1 (lstHistory)
7	Button2 (btnDeleteSelected)
8	Button3 (btnDeleteAll)
9	Button4 (btnClose)
10	Label1
11	Label2 (lblHistoryCount)

A program is needed to meet the following requirements:

When the program loads, the entire form should be visible and the cursor placed in txtNumber.

When the user clicks btnProcess, the program should validate the entry in txtNumber as being numeric and within the range of 1 to 100. Any illegal input should generate a message box displaying a suitable message, the text within the text box should be deleted and the cursor placed into the text box ready for user input.

Once the number has been validated, it should be added to lstHistory. The text box must be cleared and the cursor must be placed in the text box ready for user entry. One should be added to intHistoryCounter (see below) and that counter should be displayed in lblHistoryCount.

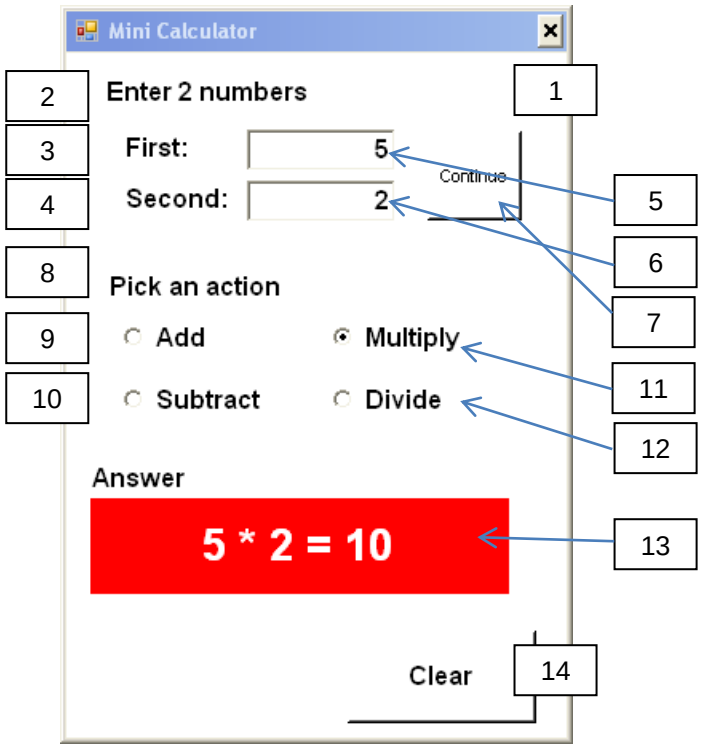
When the user clicks btnDeleteSelected, the program must make sure that the user has actually selected something in lstHistory. If they have, then that entry should be deleted. One should be subtracted from intHistoryCounter (see below) and that counter should be displayed in lblHistoryCount.

When the user clicks btnDeleteAll, the program must ask the user if they are sure that they wish to delete the entire contents of the list box via a message box. If the user answers yes, then the contents of the list box must be deleted, the contents of the text box deleted and the cursor placed in the box ready for user entry. intHistoryCounter (see below) must be set back to zero and then displayed in lblHistoryCount.

When the user clicks btnClose, the entire program should end.

The program should use an integer variable called intNumber to receive the input from the text box (after validation). There should be another integer variable called intHistoryCount which is used to keep a score of how many numbers are presently in the list box.

Exercise 4.2



Item	Object
1	Form1
2	GroupBox1
3	Label1
4	Label2
5	TextBox1 (txtFirstNumber)
6	TextBox2 (txtSecondNumber)
7	Button1 (btnContinue)
8	GroupBox2
9	RadioButton1 (radAdd)
10	RadioButton2 (radSubtract
11	RadioButton3 (radMultiply
12	RadioButton4
13	Label3 (lblAnswer)
14	Button2 (btnClear

A program is needed to meet the following requirements:

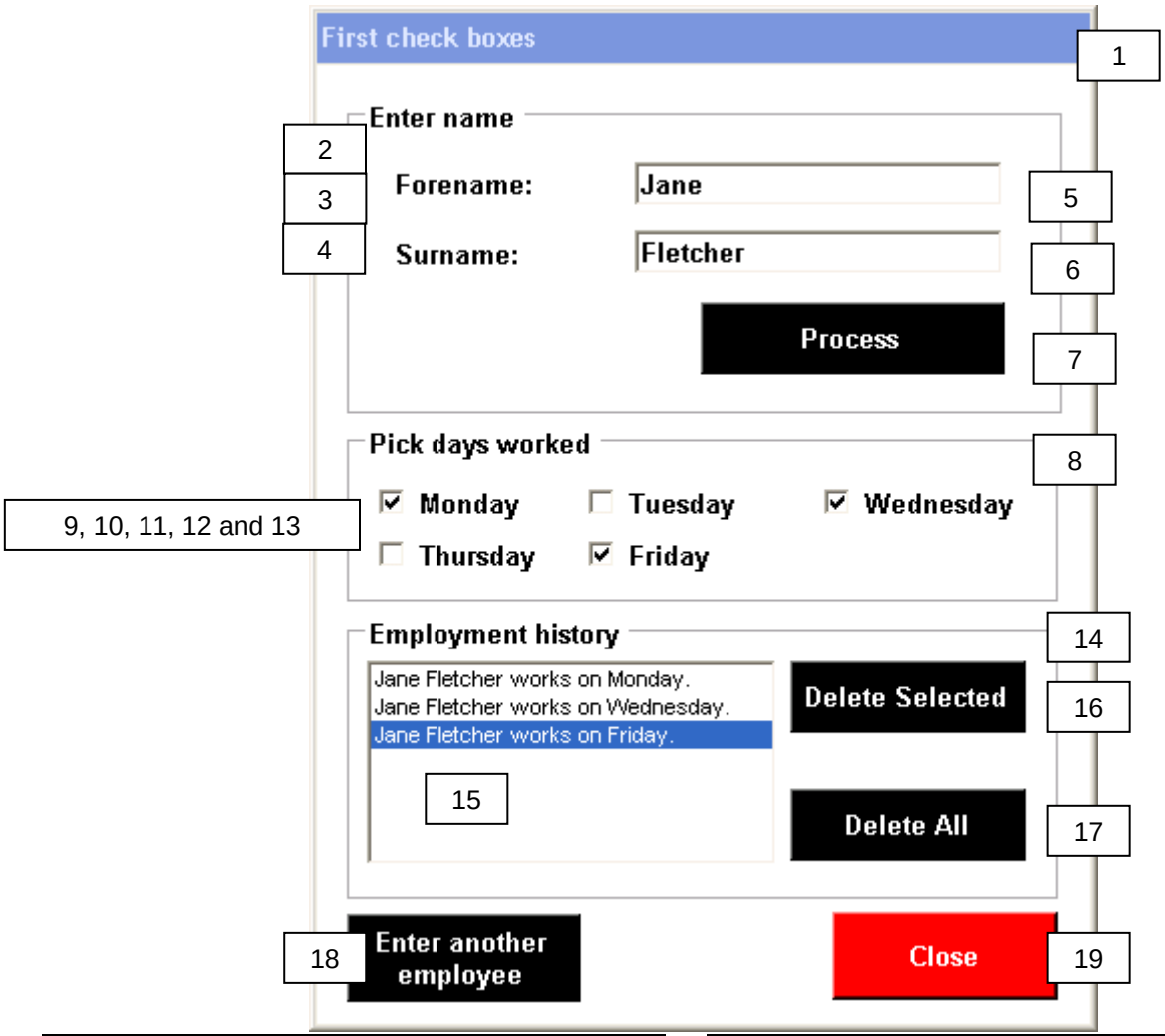
Taking the program written originally in class during week 3, try entering letters into the text boxes and clicking btnProcess. What happens?

When entering any numeric value into a program, it must always be validated. The validation necessary is as follows:

- The value must be entered (i.e. there is something there!);
- The value must not be too long (i.e. causing an overflow);
- The value must be numeric;
- The value must be in the range specified. In this instance, the numbers allowed are in the range of 1 to 100. Anything outside that range is invalid.

Remember to internally document your code.

5.0 Specification for program developed in class



Item	Object
1	Form1
2	GroupBox1
3	Label1
4	Label2
5	TextBox1 (txtForename)
6	TextBox2 (txtSurname)
7	Button1 (btnProcess)
8	GroupBox2 (gbDays)
9	CheckBox1 (chkMonday

Item	Object
10	CheckBox2 (chkTuesday)
11	CheckBox3 (chkWednesday)
12	CheckBox4 (chkThursday)
13	CheckBox5 (chkFriday)
14	GroupBox3
15	ListBox1 (lstHistory)
16	Button2 (btnDeleteSelected)
17	Button3 (btnDeleteAll)
18	Button4 (btnEnterAnotherEmployee)
19	Button5 (btnClose)

A program is needed to meet the following requirements:

When the program loads, the entire form should be visible and the cursor placed in txtForename. gbDays should be disabled.

When the user clicks btnProcess, the program should validate the entry in txtForename as s having a length between 1 and 12 characters. The same validation applies to txtSurname. Any illegal input should generate a message box displaying a suitable message, the text within the text box should be deleted and the cursor placed into the text box ready for user input.

Once the number has been validated, gbDays should be enabled and If the user checks chkMonday, the program should display a concatenated message in the format of into lstHistory:

***Forename Surname* works on Monday**

Do the same for each of the check boxes, amending the day as appropriate.

When the user clicks btnDeleteSelected, the program must make sure that the user has actually selected something in lstHistory. If they have, then that entry should be deleted.

When the user clicks btnDeleteAll, the program must ask the user if they are sure that they wish to delete the entire contents of the list box via a message box. If the user answers yes, then the contents of the list box must be deleted, the contents of the text box deleted and the cursor placed in the box ready for user entry.

When the user clicks btnEnterAnotherEmployee, gbDays should be disabled, and the text in both text boxes should be cleared. All the check boxes should be unchecked, and the cursor put in txtForename.

When the user clicks btnClose, the entire program should end.

Coding for the program

```
Public Class Form1

    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClose.Click

        'End the project
        End
    End Sub

    Private Sub btnProcess_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnProcess.Click

        'Validate the user's entry into the two text boxes to make sure
        'that they have entered something, and that that entry isn't too long.

        If txtForename.Text.Length = 0 Then
            MsgBox("You must enter a forename!", MsgBoxStyle.Critical, "ERROR!")
            txtForename.Focus()
            Exit Sub
        End If

        If txtForename.Text.Length > 12 Then
            MsgBox("You are allowed up to 12 characters for a forename!", MsgBoxStyle.Critical,
              "ERROR!")
            txtForename.Text = ""
            txtForename.Focus()
            Exit Sub
        End If

        If txtSurname.Text.Length = 0 Then
            MsgBox("You must enter a surname!", MsgBoxStyle.Critical, "ERROR!")
            txtSurname.Focus()
            Exit Sub
        End If

        If txtSurname.Text.Length > 12 Then
            MsgBox("You are allowed up to 12 characters for a surname!", MsgBoxStyle.Critical, _
              "ERROR!")
            txtSurname.Text = ""
            txtSurname.Focus()
            Exit Sub
        End If

        'Enable the group box holding the days to work check boxes.
        gbDays.Enabled = True

    End Sub

    Private Sub chkMonday_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles chkMonday.CheckedChanged

        'Enter the name and day worked into the list box.
        If chkMonday.Checked = True Then
            lstHistory.Items.Add(txtForename.Text & " " & txtSurname.Text & _
                                 " works on Monday.")
```

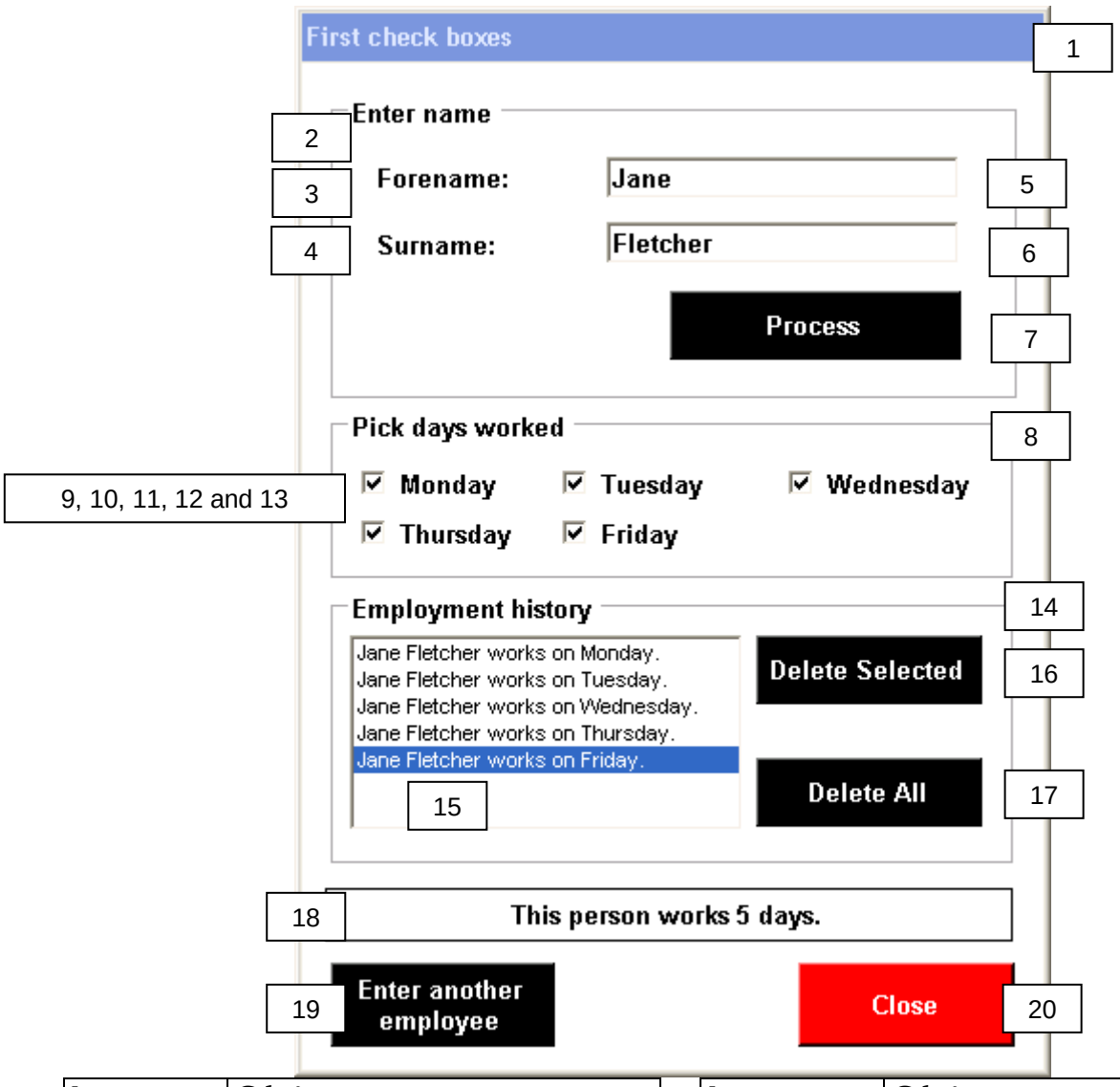
```
            SetListBox()
        End If
    End Sub

    Private Sub chkTuesday_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles chkTuesday.CheckedChanged

        'Enter the name and day worked into the list box.
        If chkTuesday.Checked = True Then
            lstHistory.Items.Add(txtForename.Text & " " & txtSurname.Text & " works on Tuesday.")
            SetListBox()
        End If
    End Sub

    Private Sub chkWednesday_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles chkWednesday.CheckedChanged

        'Enter the name and day worked into the list box.
        If chkWednesday.Checked = True Then
            lstHistory.Items.Add(txtForename.Text & " " & txtSurname.Text & " works on Wednesday.")
            SetListBox()
        End If
    End Sub

    Private Sub chkThursday_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles chkThursday.CheckedChanged

        'Enter the name and day worked into the list box.
        If chkThursday.Checked = True Then
            lstHistory.Items.Add(txtForename.Text & " " & txtSurname.Text & " works on Thursday.")
            SetListBox()
        End If
    End Sub

    Private Sub chkFriday_CheckedChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles chkFriday.CheckedChanged

        'Enter the name and day worked into the list box.
        If chkFriday.Checked = True Then
            lstHistory.Items.Add(txtForename.Text & " " & txtSurname.Text & " works on Friday.")
            SetListBox()
        End If
    End Sub

    Private Sub btnDeleteSelected_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles btnDeleteSelected.Click

        'Remove the line from the list box that the user has selected
        If lstHistory.SelectedIndex = -1 Then
            'the user hasn't selected anything.
            MsgBox("You must select a line to delete before pressing this button!", _
              MsgBoxStyle.Critical, "ERROR!")
            Exit Sub
        End If

        lstHistory.Items.RemoveAt(lstHistory.SelectedIndex)
        MsgBox("Line deleted", MsgBoxStyle.Information, "Successful deletion.")

    End Sub
```

Example programs - 5. Introducing CheckBoxes

Lecturer: Jane Fletcher

```
    Private Sub btnDeleteAll_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnDeleteAll.Click

        'Ask the user if they really mean to delete everything.
        Dim intResponse As Integer

        intResponse = MsgBox("Do you really wish to delete everything?", _
                             MsgBoxStyle.YesNo, "Are you sure?")

        If intResponse = vbYes Then
            'they do.
            lstHistory.Items.Clear()
            txtForename.Text = ""
            txtSurname.Text = ""
            txtForename.Focus()
            MsgBox("Information cleared.", MsgBoxStyle.Information, _
                   "All information deleted.")
        Else
            MsgBox("No information deleted.", MsgBoxStyle.Information, _
                   "No action taken.")
        End If
    End Sub

    Private Sub btnEnterAnotherEmployee_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles btnEnterAnotherEmployee.Click

        'Reset everything back to the beginning
        txtForename.Text = ""
        txtSurname.Text = ""
        gbDays.Enabled = False
        chkMonday.Checked = False
        chkTuesday.Checked = False
        chkWednesday.Checked = False
        chkThursday.Checked = False
        chkFriday.Checked = False
        txtForename.Focus()
    End Sub

    Private Sub SetListBox()
        Dim intTempCount As Integer = lstHistory.Items.Count
        lstHistory.SelectedIndex = (intTempCount - 1)
    End Sub

End Class
```

Exercise 5.1



Item	Object
1	Form1
2	GroupBox1
3	Label1
4	Label2
5	TextBox1 (txtForename)
6	TextBox2 (txtSurname)
7	Button1 (btnProcess)
8	GroupBox2 (gbDays)
9	CheckBox1 (chkMonday)
10	CheckBox2 (chkTuesday)

Item	Object
11	CheckBox3 (chkWednesday)
12	CheckBox4 (chkThursday)
13	CheckBox5 (chkFriday)
14	GroupBox3
15	ListBox1 (lstHistory)
16	Button2 (btnDeleteSelected)
17	Button3 (btnDeleteAll)
18	Label3 (lblCount)
19	Button4 (btnEnterAnotherEmployee)
20	Button5 (btnClose)

A program is needed to meet the following requirements:

An integer variable intCounter should be declared at the top of the program.

When the program loads, the entire form should be visible and the cursor placed in txtForename. gbDays should be disabled.

When the user clicks btnProcess, the program should validate the entry in txtForename as s having a length between 1 and 12 characters. The same validation applies to txtSurname. Any illegal input should generate a message box displaying a suitable message, the text within the text box should be deleted and the cursor placed into the text box ready for user input.

Once the number has been validated, gbDays should be enabled and If the user checks chkMonday, the program should display a concatenated message in the format of into lstHistory:

***Forename Surname* works on Monday**

Do the same for each of the check boxes, amending the day as appropriate. Add one to a count of days worked (intCounter), and display this in lblCount in the format shown here:

"This person works " *intCounter* " days."

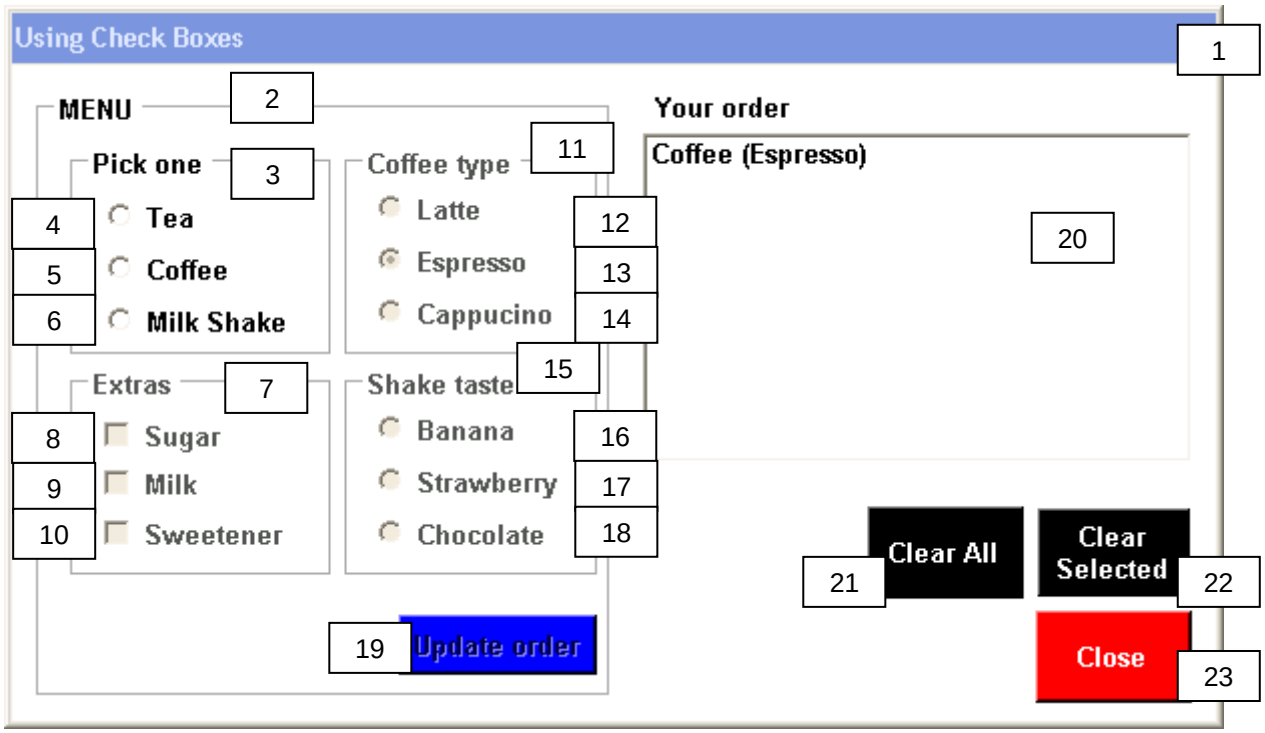
When the user clicks btnDeleteSelected, the program must make sure that the user has actually selected something in lstHistory. If they have, then that entry should be deleted. One should be removed from the variable intCounter, and the new total of days worked updated in lblCount.

When the user clicks btnDeleteAll, the program must ask the user if they are sure that they wish to delete the entire contents of the list box via a message box. If the user answers yes, then the contents of the list box must be deleted, the contents of the text box deleted and the cursor placed in the box ready for user entry. lblCount should be cleared, and the variable intCounter set to equal 0.

When the user clicks btnEnterAnotherEmployee, gbDays should be disabled, and the text in both text boxes should be cleared. lblCount should be cleared, and the variable intCounter set to equal 0. All the check boxes should be unchecked, and the cursor put in txtForename.

When the user clicks btnClose, the entire program should end.

Exercise 5.2



Item	Object
1	Form1
2	GroupBox1
3	GroupBox2
4	RadioButton1 (radTea)
5	RadioButton2 (radCoffee)
6	RadioButton3 (radMilkShake)
7	GroupBox3 (gbExtras)
8	CheckBox1 (chkSugar)
9	CheckBox2 (chkMilk)
10	CheckBox3 (chkSweetener)
11	GroupBox4 (gbCoffeeType)

Item	Object
12	CheckBox4 (chkLatte)
13	CheckBox5 (chkEspresso)
14	CheckBox6 (chkCappucino)
15	GroupBox5 (gbMSFlavour)
16	RadioButton4 (radBanana)
17	RadioButton5 (radStrawberry)
18	RadioButton6 (radChocolate)
19	Button1 (btnUpdateorder)
20	ListBox1 (lstYourOrder)
21	Button2 (btnClearAll)
22	Button3 (btnClearSelected)
23	Button4 (btnClose)

A program is needed to meet the following requirements:

General
String variables strDrink, strType, strSugar, strMilk, strOrder and strSweetner should be declared at the top of the program.

When the program loads, all group boxes except GroupBox1 and GroupBox2 should be disabled. Note that neither of these are referred to within the code and therefore do not have to have a name other than the default allocated by Visual Basic .NET. Once the user has selected a choice of beverage, other group boxes will become available as appropriate. btnUpdateOrder should be disabled.

Main Drinks menu
If the user selects tea, then gbExtras should be enabled. Put the value "Tea " into the variable strDrink.

If the user selects coffee, then gbCoffeeType should be enabled. Put the value "Coffee " into the variable strDrink.

If the user selects milkshake, then gbMSFlavour should be enabled. Put the value "Milkshake " into the variable strDrink.

Once a drink has been selected, btnUpdateOrder should be enabled.

NOTE THE EXTRA SPACE AFTER THE TYPE OF DRINK IN EACH CASE.

Extras menu
If the user chooses to have sugar with their drink, then put the value "sugar " into the variable strSugar.

If the user chooses to have milk with their drink, then put the value "milk " into the variable strMilk.

If the user chooses to have sweetener with their drink, then put the value "sweetener " into the variable strSweetener.

NOTE THE EXTRA SPACE AFTER THE SELECTED EXTRA IN EACH CASE.

Coffee Type menu
If the user selects latte, put "(Latte) " into the variable strType. The check box for milk should be enabled.

If the user selects espresso, put "(Espresso) " into the variable strType. The check box for milk should be disabled and unchecked, and the variable strMilk should be set to nothing.

If the user selects cappucino, put "(Cappucino) " into the variable strType. The check box for milk should be enabled.

NOTE THE EXTRA SPACE AFTER THE SELECTED TYPE OF COFFEE IN EACH CASE.

Shake taste menu
If the user selects banana, put "(Banana) " into the variable strType.

If the user selects strawberry, put "(Strawberry) " into the variable strType.

If the user selects chocolate, put "(Chocolate) " into the variable strType.

NOTE THE EXTRA SPACE AFTER THE SELECTED FLAVOUR IN EACH CASE.

btnUpdateOrder
This button will build up the type of drink ordered and all flavours / extras etc and display it in the list box lstYourOrder. It will then clear everything down so that a new order can be made.

Once the user clicks btnUpdateOrder, move the value that is in strDrink into strOrder, like so: strOrder = strDrink.

If chkMilk is selected, concatenate the value that is in strOrder with the value in strMilk, like so: strOrder = strOrder & "with " & strMilk.

If chkSugar is selected, concatenate the value that is in strOrder with the value in strSugar, using the same technique as for chkMilk.

If chkSweetener is selected, concatenate the value that is in strOrder with the value in strSweetener.

Add the value contained in strOrder into lstYourOrder.

Then:

Enable chkMilk, disable gbCoffeeType, gbExtras and gbMSFlavours. Uncheck chkMilk, chkSugar and chkSweetener. Disable btnUpdateOrder.

btnClearSelected
When the user clicks btnClearSelected, the program must make sure that the user has actually selected something in lstYourOrder. If they have, then that entry should be deleted.

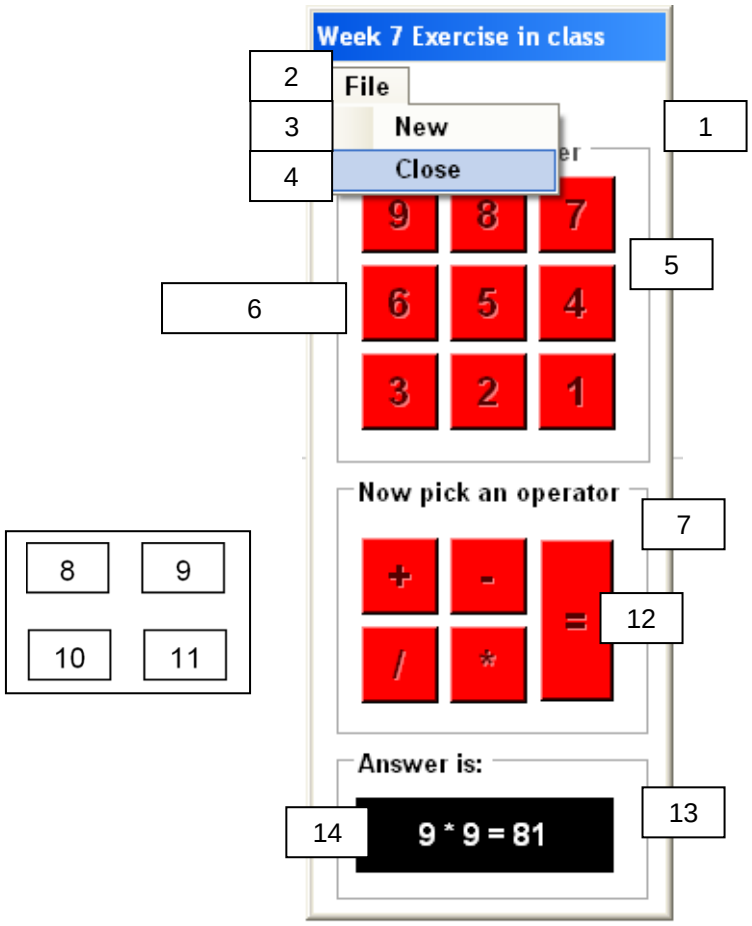
btnClearAll
When the user clicks btnClearAll, the program must ask the user if they are sure that they wish to delete the entire contents of the list box via a message box. If the user answers yes, then the contents of the list box must be deleted.

Then:

Enable chkMilk, disable gbCoffeeType, gbExtras and gbMSFlavours. Uncheck chkMilk, chkSugar and chkSweetener. Disable btnUpdateOrder.

btnClose
When the user clicks btnClose, the entire program should end.

6.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	GroupBox1 (gbNumbers)
6	Button1 – Button9 (btn1-btn9)
7	GroupBox2 (gbOperator)
8	Button10 (btnPlus)
9	Button11 (btnMinus)

Item	Object
10	Button12 (btnDivide)
11	Button13 (btnMultiply)
12	Button14 (btnEquals)
13	GroupBox3 (gbAnswer)
14	Label1 (lblAnswer)

A program is needed to meet the following requirements:

Overview

1. When the program loads, only gbNumbers should be visible, with the height of the form adjusted accordingly.
2. The program needs these variables: two integers: intFirst and intSecond; a decimal decAnswer; a Boolean: bFirst; and a string: strOperator. These should all be declared at the top of the program.
3. The variable bFirst should be set to True.
4. The background colour of the menu strip should be set to be the same as the background colour of the form.
5. The font of the menu strip should be the same as the font on the form.
6. The menu strip should have one item at the top level: File.
7. File should contain two items: New and Exit.
8. Exit should close the program.
9. New should reset the form to its original condition; the height should be adjusted and all objects that have had their Enabled property altered during the execution of the program should be reset.

Using the program

10. Once the user has clicked on a "number" button, the group box gbNumbers should be disabled.
11. The value of the variable bFirst should be set to False to indicate that the first number has now been entered.
12. The value of the text of the selected button should be stored in intFirst.
13. The height of the form should be adjusted to show gbOperator.
14. btnEquals should be disabled at this point so that the user may select either plus, minus, multiply or divide.
15. Once the selection has been made by the user clicking his or her chosen button, the height of the form should be altered so that only gbNumbers is visible.
16. The text of the button selected should be stored in the variable strOperator.
17. gbNumbers must be made accessible again.
18. Once the user has clicked his second number, then the height of the form should be adjusted to again show gbOperator.
19. The value of the text of the selected button should be stored in intSecond.
20. Now, all buttons in that group box except btnEquals should be disabled.
21. When btnEquals has been selected, the height of the form should be adjusted to show gbAnswer.
22. The value of decAnswer should be determined by the nature of the value stored in strOperator, using the Select Case method.
23. The answer should be displayed in lblAnswer in the following format:

first number selected	*operator*	*second number selected*	*equals*	*answer*
1	+	*3*	=	*4*

24. All group boxes must be disabled at this point so that the only option open to the user is to select the File menu.

Behind the scenes

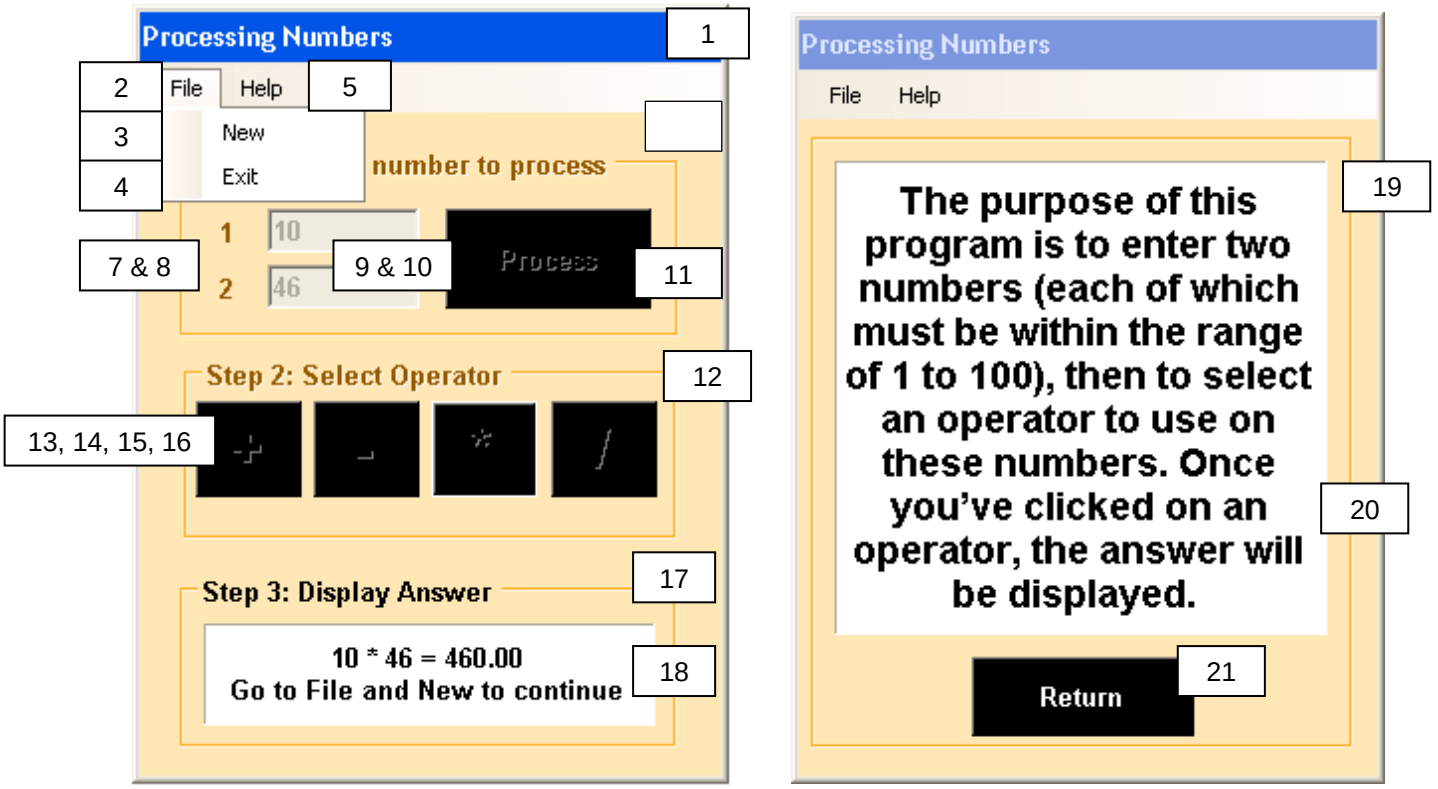
25. For each of the "number" buttons, the coding is the same so the one click event is to be used for all of them. Use the "sender.text" method of ascertaining which number has been clicked, and store the sender.text in intFirst.

Coding for the program

```
Public Class Form1

    Dim bFirst As Boolean
    Dim intFirst, intSecond As Integer
    Dim decAnswer As Decimal
    Dim strOperator As String

    Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click

        'Close the project
        End
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        'Alter the colour of the menu strip
        MenuStrip1.BackColor = Color.White

        'Set the height of the form.
        Me.Height = 248

        'Tell the program that it is the first number that the user is picking
        bFirst = True
    End Sub


    Private Sub btn9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btn9.Click, btn8.Click, btn7.Click, btn6.Click, btn5.Click, _
              btn4.Click, btn3.Click, btn2.Click, btn1.Click

        'if bFirst is true then it means that we're in the position of picking
        'out the first number.

        'sender is the word for the actual button that the user clicks. Since we're doing the same
        'processing for all the buttons, we don't need to write a separate event for each of them.
        'The same code does for all.

        'The only difference in processing is whether we're on the first number or the second one.
        If bFirst Then
            intFirst = sender.text
            bFirst = False
        Else
            intSecond = sender.text
        End If

        'Alter the size of the form
        Me.Height = 391
        gbNumbers.Enabled = False
    End Sub


    Private Sub btnPlus_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnPlus.Click,
        btnMinus.Click, btnMultiply.Click, btnDivide.Click
```

```
        'Save the operator that the user wants to use and then disable all operators but equals
        strOperator = sender.text
        gbNumbers.Enabled = True
        btnEquals.Enabled = True
        btnDivide.Enabled = False
        btnPlus.Enabled = False
        btnMultiply.Enabled = False
        btnMinus.Enabled = False
        Me.Height = 248

    End Sub

    Private Sub btnEquals_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnEquals.Click

        'Here we're going to use a select case statement rather than a string of If statements.
        'Select Case is a form of selection construct. It examines whatever you tell the program
        'to use as a determinant, and then acts accordingly.

        'Work out the answer to the sum
        Select Case strOperator
            Case "+"        'The user picked out to add in the operator bit.
                decAnswer = intFirst + intSecond
            Case "-"        'The user picked to subtract in the operator bit.
                decAnswer = intFirst + intSecond
            Case "/"        'The user picked to divide in the operator bit.
                decAnswer = intFirst + intSecond
            Case "*"        'The user picked to multiply in the operator bit.
                decAnswer = intFirst * intSecond
        End Select

        'Concatenate an answer for the label.
        lblAnswer.Text = intFirst & " " & strOperator & " " & intSecond & " = " & decAnswer

        'disable the equals button
        btnEquals.Enabled = False

        'Alter the height of the form
        Me.Height = 476

    End Sub

    Private Sub NewToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles NewToolStripMenuItem.Click

        'Put everything back to its original state
        Me.Height = 248 'Change the height of the form
        bFirst = True   'We're starting again from scratch

        gbNumbers.Enabled = True 'So that the user can start picking out numbers again

        'Enable all the operator buttons and disable equals
        btnDivide.Enabled = True
        btnPlus.Enabled = True
        btnMultiply.Enabled = True
        btnMinus.Enabled = True
        btnEquals.Enabled = False
    End Sub
End Class
```

Exercise 6.1



Item	Object
1	Form1
2	MenuStrip1 - File
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	HelpToolStripMenuItem
6	GroupBox1 (gbNumbers)
7	Label1
8	Label2
9	TextBox1 (txtNumber1)
10	TextBox2 (txtNumber2)
11	Button1 (btnProcess)

Item	Object
12	GroupBox2 (gbOperator)
13	Buton2 (btnAdd)
14	Button3 (btnMinus)
15	Button4 (btnMultiply)
16	Button5 (btnDivide)
17	GroupBox3
18	Label3 (lblAnswer)
19	GroupBox4 (gbAbout)
20	Label4
21	Button6 (btnReturn)

Example programs – 6. MenuStrips

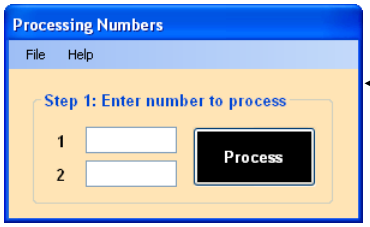
Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

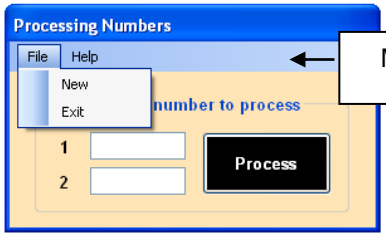
A program is required which will allow the user to enter two numbers, each of which should be within the range of 1 to 100. After the numbers have been entered and proved valid, the user should be allowed to select a mathematical operator (+, -, * or /) and the resulting calculation should be displayed. Variables intNumber1, intNumber2, intSaveHeight and decTotal should be declared at the top of the program.

An "About" feature is also required to tell the user how the program functions.

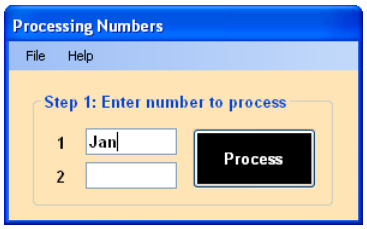
The program should be internally documented in full.

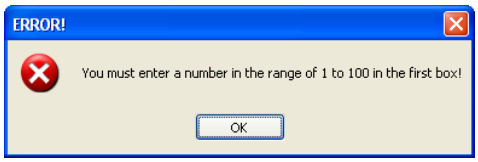


Start-up screen

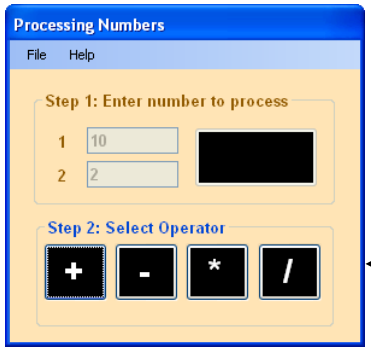


Menu bar

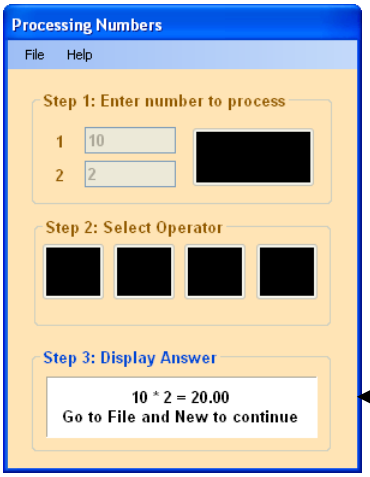




Example validation



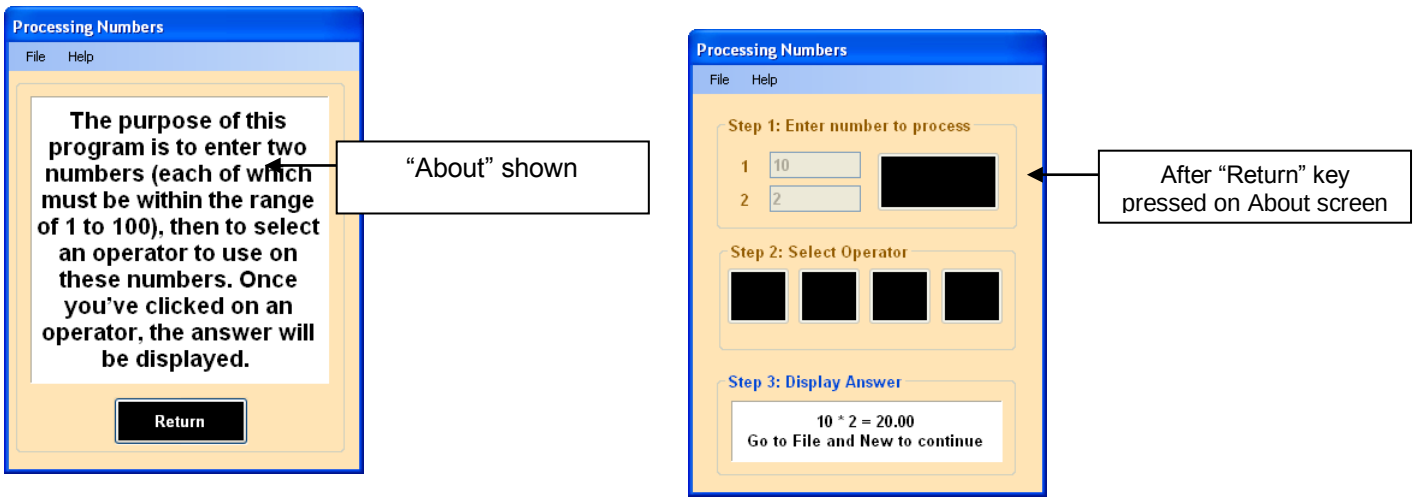
Valid entry, first GB disabled, operators shown, form enlarged



Operator (*) selected, second GB disabled, form enlarged, answer shown. Instructions on how to continue shown.

Example programs – 6. MenuStrips

Lecturer: Jane Fletcher



Instead of from a specification, write the project above from the structured English given.

Structured English for this program

Form_Load

Set the height of Form1 to be 182
Set focus to txtNumber1

End of Form_Load

---

btnProcess_Click

If the length of the entry in txtNumber1 = 0 Then
    Display a message box saying "You must enter a number in the first box!"
    Set the cursor focus to txtNumber1
    Quit the subroutine
End If

If the length of the entry in txtNumber1 is > 3 Then
    Display a messagebox saying "You must enter a number in the range of 1 to 100 in the first box"
    Blank out the text in txtNumber1
    Set the cursor focus to txtNumber1
    Quit the subroutine
End If

If the entry in txtNumber1 isn't a number
    Display a messagebox saying "You must enter a number in the range of 1 to 100 in the first box "
    Blank out the text in txtNumber1
    Set the cursor focus to txtNumber1
    Quit the subroutine
End If

Set intNumber1 to hold the value contained in txtNumber1

Check the range entered by comparing intNumber1 with 100 and 1 –
if it is outside that range then
    Display a messagebox saying "You must enter a number in the range of 1 to 100 in the first box"
    Blank out the text in txtNumber1

```
            Set the cursor focus to txtNumber1
            Quit the subroutine
    End If

    Check that there is no decimal place contained in txtNumber1
    if there is then
            Display a messagebox saying "You must enter an integer in the range of 1 to 100 in the first box”
            Blank out the text in txtNumber1
            Set the cursor focus to txtNumber1
            Quit the subroutine
    End If

    If the length of the entry in txtNumber2  = 0 Then
            Display a message box  saying "You must enter a number in the second box!"
            Set the cursor focus to txtNumber2
            Quit the subroutine
    End If

    If the length of the entry in txtNumber2 is > 3 Then
            Display a messagebox saying "You must enter a number in the range of 1 to 100 in the second box”
            Blank out the text in txtNumber2
            Set the cursor focus to txtNumber2
            Quit the subroutine
    End If

    If the entry in txtNumber2 isn’t a number
            Display a messagebox saying "You must enter a number in the range of 1 to 100 in the second box”
            Blank out the text in txtNumber2
            Set the cursor focus to txtNumber2
            Quit the subroutine
    End If

    Set  intNumber2 to hold the value contained in txtNumber2

    Check the range entered by comparing intNumber2 with 100 and 1 –
    if it is outside that range then
            Display a messagebox saying "You must enter a number in the range of 1 to 100 in the second box”
            Blank out the text in txtNumber2
            Set the cursor focus to txtNumber2
            Quit the subroutine
    End If

    Check that there is no decimal place contained in txtNumber2
    if there is then
            Display a messagebox saying "You must enter an integer in the range of 1 to 100 in the second box”
            Blank out the text in txtNumber2
            Set the cursor focus to txtNumber2
            Quit the subroutine
    End If

    Set the height of the form to be 284
    Bring the group box gbOperator to the front
    Disable the group box gbNumbers
```

End of btnProcess_Click

Click event for File-> Exit

Dispose of the form
End the project

End of FileExit_Click

Click event for File-> New

Perform the Subroutine ClearStuff

End of FileNew_Click

Subroutine ClearStuff

Set the Text properties of txtNumber1 and txtNumber2 to equal “”
Set the Text property of lblAnswer to equal “”
Set the values in intNumber1, intNumber2 and decTotal to equal 0
Set the height of the form to equal 182
Enable the GroupBox gbNumbers
Enable the GroupBox gbOperator
Set the cursor focus to be on txtNumber1

End of ClearStuff

Click event for Help

Bring to the front gbAbout
Set gbAbout to be visible
Save the height of the form in its present state into intSaveHeight
Set the height of the form to equal 389

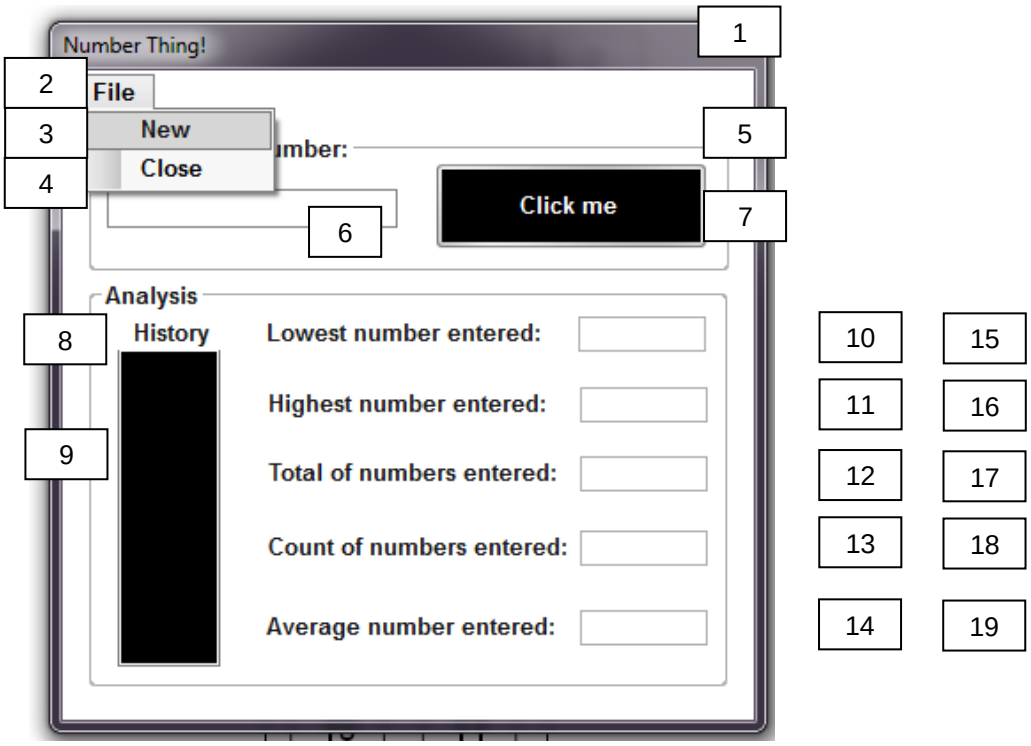
End of Help_Click

Click event for btnReturn

Send gbAbout to the back
Set gbAbout to be invisible
Set the height of the form to equal the value stored in intSaveHeight

End of btnReturn_Click

7.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	GroupBox1
6	TextBox1 (txtNumber)
7	Button1 (btnClickMe)
8	GroupBox2
9	ListBox1 (lstNumbers)
10	Label1

Item	Object
11	Label2
12	Label3
13	Label4
14	Label5
15	Label6 (lblLowest)
16	Label7 (lblHighest)
17	Label8 (lblTotal)
18	Label9 (lblCount)
19	Label10 (lblAverage)

A program is needed to meet the following requirements:

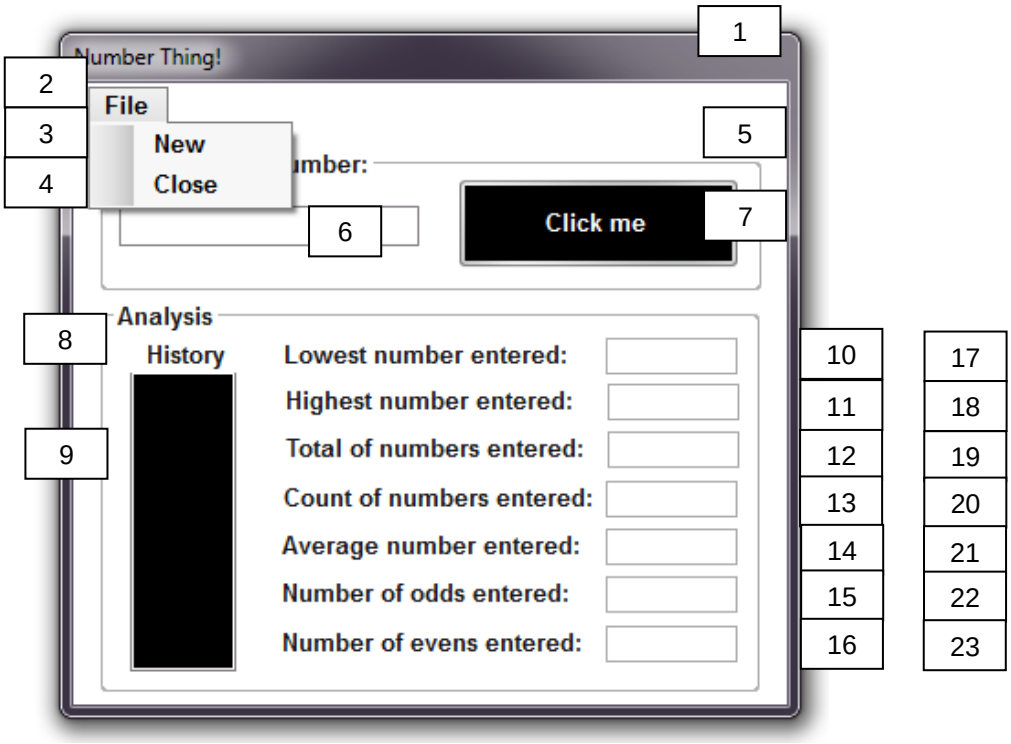
1. The menu strip should have one item at the top level: File.
2. File should contain two items: New and Exit.
3. Exit should close the program.
4. New should reset the form to its original condition and all variables should be set to their original values.
5. When the program loads, everything should be visible on the form.
6. The program needs these variables: five integers: intNumber, intCount, intTotal, intHighest and intLowest; and a decimal decAverage. These should all be declared at the top of the program.
7. The value 0 should be stored in intHighest.
8. The value 10,000 should be stored in intLowest.
9. The cursor should be set into the text box.
10. The user's entry should be validated as being numeric and in the range of 1 to 9999. Any invalid entry should be reported to the user via a message box, invalid input should be removed and the cursor set into the text box ready for further user entry.
11. Valid user entry should be stored in the variable intNumber.
12. Each number that is entered by the user should be processed as follows:
    a. If the value in intNumber is less than the value in intLowest, then store the value of intNumber into the variable intLowest and display this number in lblLowest.
    b. If the value in intNumber is greater than the value in intHighest, then store the value of intNumber into the variable intHighest and display this number in lblHighest.
    c. intTotal should be added to the value in variable intTotal and that value displayed in the label lblTotal.
    d. One should be added to the variable intCount.
    e. decAverage should be calculated by dividing intTotal by intCount, and this value stored in decAverage.
13. The text box txtNumber should be cleared and the cursor replaced there to wait for user input after this processing has occurred.

Coding for this project

```
Public Class Form1

    Dim intNumber As Integer

    Dim intHighest As Integer = 0

    Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click

        'End the project
        End
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        'Change the properties of the menustrip
        MenuStrip1.BackColor = Color.White
        MenuStrip1.ForeColor = Color.Black
    End Sub

    Private Sub btnClickMe_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClickMe.Click

        'Validate the number entered

        If txtNumber.Text.Length = 0 Then
            MsgBox("You must enter a number!", MsgBoxStyle.Critical, "No input!")
            txtNumber.Focus()
            Exit Sub
        End If

        If txtNumber.Text.Length > 4 Then
            MsgBox("You have entered too much information!", MsgBoxStyle.Critical, _
              "Too much input!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtNumber.Text) Then
            MsgBox("You must enter a number in the range of 1 to 9999!", MsgBoxStyle.Critical, _
              "You must enter a number in the right range!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        intNumber = txtNumber.Text

        If (intNumber < 1) Or (intNumber > 9999) Then
            MsgBox("You must enter a number in the range of 1 to 9999!", MsgBoxStyle.Critical, _
              "You must enter a number in the right range!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
```

```
        End If

        ProcessNumber()

        'Put the cursor back into the text box after clearing it.
        txtNumber.Text = ""
        txtNumber.Focus()
    End Sub

    Private Sub ProcessNumber()

        'Process the number entered.

        If intNumber > intHighest Then
            'It is so store it as the current highest number entered.
            intHighest = intNumber
        End If

        'Now display all new values in the label.
         lblHighest.Text = intHighest
    End Sub

    Private Sub NewToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles NewToolStripMenuItem.Click

        'Clear everything down and set back to original values.
        intHighest = 0
        txtNumber.Text = ""
        txtNumber.Focus()
    End Sub
End Class
```

Exercise 7.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	GroupBox1
6	TextBox1 (txtNumber)
7	Button1 (btnClickMe)
8	GroupBox2
9	ListBox1 (lstNumbers)
10	Label1
11	Label2
12	Label3

Item	Object
13	Label4
14	Label5
15	Label6
16	Label7
17	Label8 (lblLowest)
18	Label9 (lblHighest)
19	Label10 (lblTotal)
20	Label11 (lblCount)
21	Label12 (lblAverage)
22	Label13 (lblOdds)
23	Label14 (lblEvens)

A program is needed to meet the following requirements:

1. The menu strip should have one item at the top level: File.
2. File should contain two items: New and Exit.
3. Exit should close the program.
4. New should reset the form to its original condition and all variables should be set to their original values.
5. When the program loads, everything should be visible on the form.
6. The program needs these variables: seven integers: intNumber, intCount, intTotal, intHighest,intLowest, intOdds and intEvens; and a decimal decAverage. These should all be declared at the top of the program.
7. The value 0 should be stored in intHighest.
8. The value 10,000 should be stored in intLowest.
9. The cursor should be set into the text box.
10. The user's entry should be validated as being numeric and in the range of 1 to 9999. Any invalid entry should be reported to the user via a message box, invalid input should be removed and the cursor set into the text box ready for further user entry.
11. Valid user entry should be stored in the variable intNumber.
12. Each number that is entered by the user should be processed as follows:
    a. If the value in intNumber is less than the value in intLowest, then store the value of intNumber into the variable intLowest and display this number in lblLowest.
    b. If the value in intNumber is greater than the value in intHighest, then store the value of intNumber into the variable intHighest and display this number in lblHighest.
    c. intTotal should be added to the value in variable intTotal and that value displayed in the label lblTotal.
    d. One should be added to the variable intCount.
    e. decAverage should be calculated by dividing intTotal by intCount, and this value stored in decAverage.
    f. If the number entered is odd, then one should be added to the variable intOdds and that value stored in lblOdds.
    g. If the number entered is even, then one should be added to the variable intEvens and that value stored in lblEvens.
13. The text box txtNumber should be cleared and the cursor replaced there to wait for user input after this processing has occurred.

8.0 Specification for program developed in class

Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	Label1 (lblTodaysDate)

Item	Object
6	GroupBox1
7	DateTimePicker1 (dtpSelectedDate)
8	GroupBox2
9	Label2 (lblMessage)

A program is needed to meet the following requirements:

Overview

1. When the program loads, the entire form should be visible.
2. Throughout the program use the font of your choice.
3. Use the background image of your choice for the form. Set the BackgroundImageLayout of the form to be "Stretch" (if appropriate to your image).
4. The program needs these variables declared at the top of the program:

```
Dim dtToday As Date = Date.Today
Dim dtSelectedDate As Date
Dim intYear As Integer = dtToday.Year
Dim dtChristmas As Date = New Date(intYear, 12, 25)
Dim intDay As Integer
Dim strDay As String
```

5. The font of the menu strip should be the same as the font on the form.
6. The menu strip should have one item at the top level: **File**.
7. File should contain two items: **New** and **Exit**.
8. **Exit** should close the program.
9. **New** should reset the form to its original condition; the label lblMessage should be cleared of its text and the date of the datetimepicker dtpSelectedDate should be set to today's date.
10. Upon the Form_Load event, set the background colour of the menustrip to be white, and the foreground to be black.
11. Also on the Form_Load event, the program should work out the day of the week held in the current system date. This value should be stored in the variable intDay.

```
intDay = Date.Today.DayOfWeek
```

12. The correct name of the day should be determined by doing a Select..Case operation on the value contained in intDay. If intDay contains 1, then the day is Monday – and so on. The result of the Select..Case should be stored in strDay.
13. Once the day has been established, the present date should be displayed in the label lblTodaysDate in the format **strDay & ", " & Date.Today**.
14. The CloseUp event of a DateTimePicker control is activated when the DateTimePicker has been used and subsequently "closes up". On the CloseUp event of dtpSelectedDate the date value of the DateTimePicker should be stored in dtSelectedDate.
15. The difference between the selected date and Christmas of the current year should be determined by using a DateDiff operation, in the following format:

```
intDaysDifference = DateDiff(DateInterval.Day, dtSelectedDate, dtChristmas)
```

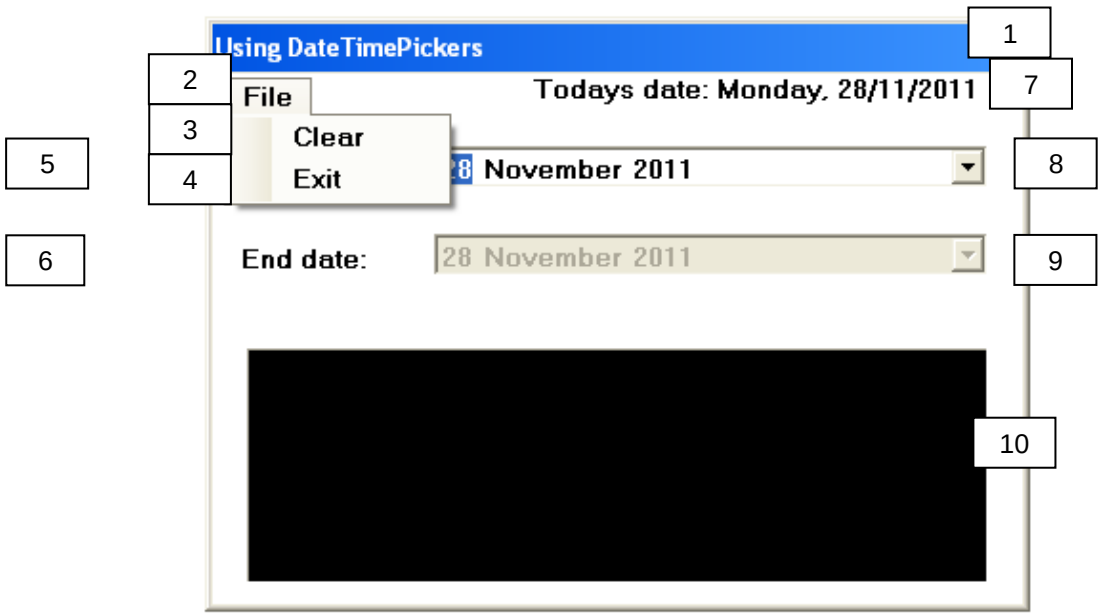
    where intDaysDifference is declared as a local integer. This operation takes the "Day" interval as its required result, and works out the difference between the two dates.
16. Display the days to Christmas from the selected date in lblMessage, in the format "*dtSelectedDate* is *intDaysDifference* to Christmas".

Coding for this program

```
Public Class Form1

    Dim dtToday As Date = Date.Today
    Dim dtSelectedDate As Date
    Dim intYear As Integer = dtToday.Year
    Dim dtChristmas As Date = New Date(intYear, 12, 25)
    Dim intDay As Integer
    Dim strDay As String

    Private Sub CloseToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
              Handles CloseToolStripMenuItem.Click

        'End the project
        End
    End Sub

    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

        'Set the defaults for the form
        MenuStrip1.ForeColor = Color.Black
        MenuStrip1.BackColor = Color.White

        'The following code is to work out what day it is today.

        intDay = Date.Today.DayOfWeek
        'Generates an integer value - 1 for Monday, 2 for Tuesday and so on

        Select Case intDay
            Case 1
                strDay = "Monday"
            Case 2
                strDay = "Tuesday"
            Case 3
                strDay = "Wednesday"
            Case 4
                strDay = "Thursday"
            Case 5
                strDay = "Friday"
            Case 6
                strDay = "Saturday"
            Case 7
                strDay = "Sunday"
        End Select

        'Display the complete date in the label
        lblTodaysDate.Text = strDay & ", " & Date.Today

    End Sub

    Private Sub dtpSelectedDate_CloseUp(ByVal sender As Object, ByVal e As System.EventArgs) _
              Handles dtpSelectedDate.CloseUp

        'The default event for a date time picker is the "ValueChanged" event.
        'The CloseUp event means that the user has used the DTP and has now moved on from it.
```

```
        'Assign the value of the date time picker to the date variable dtSelectedDate
        dtSelectedDate = dtpSelectedDate.Value.Date

        'Work out the difference between the date the user has picked and Christmas day
        'using days as the outcome (not months, or years)
        Dim intDaysDifference As Integer
        intDaysDifference = DateDiff(DateInterval.Day, dtSelectedDate, dtChristmas)

        If intDaysDifference = 0 Then
            lblMessage.Text = "It's CHRISTMAS!"
        ElseIf intDaysDifference < 1 Then
            lblMessage.Text = "It's " & (intDaysDifference * -1) & " days since Christmas!"
        Else
            lblMessage.Text = dtSelectedDate & " is " & intDaysDifference & " days to Christmas!"
        End If

    End Sub

    Private Sub NewToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
              Handles NewToolStripMenuItem.Click

        'Get rid of the contents of the label
        lblMessage.Text = ""

        'Reset the datetimepicker to be today's date
        dtpSelectedDate.Value = Date.Today
    End Sub
End Class
```

Exercise 8.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	ClearToolStripMenuItem
4	ExitToolStripMenuItem
5	Label1

Item	Object
6	Label2
7	Label3(lblTodaysDate)
8	DateTimePicker1 (dtpStartDate)
9	DateTimePicker2 (dtpEndDate)
10	Label4 (lblMessage)

Example programs – 8. DateTimePickers

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

Overview

1. When the program loads, the entire form should be visible.
2. Throughout the program use the font of your choice.
3. The program needs these variables declared at the top of the program:

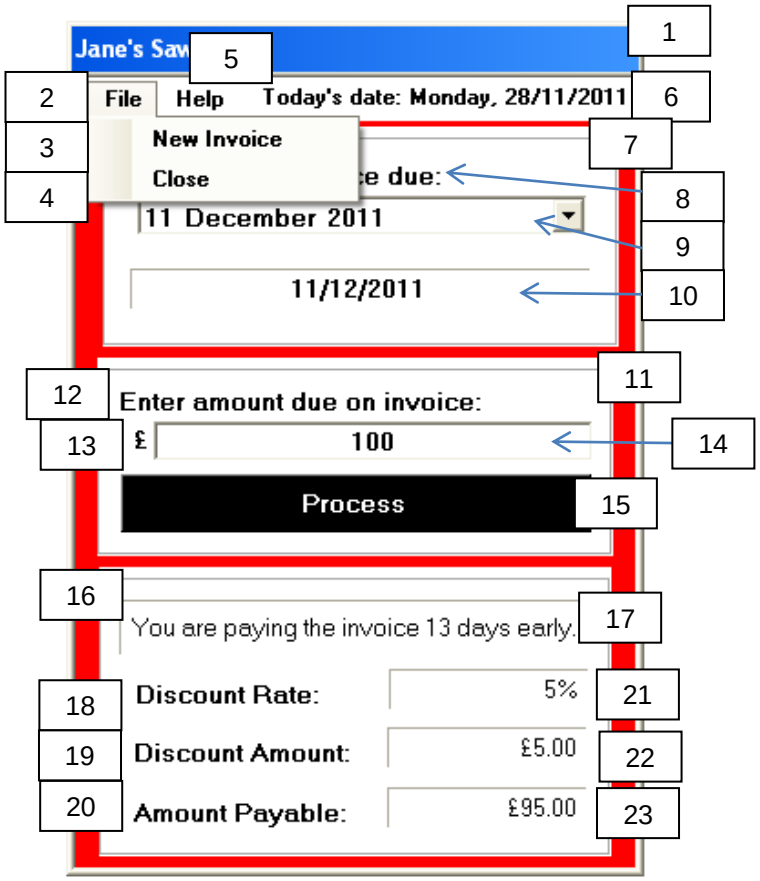
```
Dim dtTodaysDate As Date = Date.Today
Dim intDay As Integer = dtTodaysDate.DayOfWeek
Dim dtStartDate, dtEndDate As Date
Dim intDaysDifference As Integer
```

4. The font of the menu strip should be the same as the font on the form.
5. The menu strip should have one item at the top level: **File**.
6. File should contain two items: **Clear** and **Exit**.
7. **Exit** should close the program.
8. **Clear** should reset the form to its original condition; the label lblMessage should be cleared of its text and the date of the two datetimepickers should be set to today's date. dtpEndDate should be disabled.
9. Upon the Form_Load event, set the background colour of the menustrip to be white, and the foreground to be black.
10. Upon Form_Load, dtpEndDate should be disabled.
11. On the Form_Load event, the correct name of the day should be determined by doing a Select..Case operation on the value contained in intDay. If intDay contains 1, then the day is Monday – and so on. The result of the Select..Case should be stored in strDay.
12. Once the day has been established, the present date should be displayed in the label lblTodaysDate in the format **strDay & ", " & Date.Today**.
13. On the CloseUp event of dtpStartDate the date value of the DateTimePicker should be stored in dtStartDate, dtpStartDate should be disabled and dtpEndDate should be enabled.
14. The difference between the selected start date and the selected end date should be determined by using a DateDiff operation, in the following format:

```
intDaysDifference = DateDiff(DateInterval.Day, dtStartDate, dtEndDate)
```

    where intDaysDifference is declared as a local integer.
15. Display the days to Christmas from the selected date in lblMessage, in the format "The difference between these two dates is …." as appropriate.

Exercise 8.2



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	HelpToolStripMenuItem
6	Label1 (lblDate)
7	GroupBox1 (gbInvoiceDate)
8	Label2
9	DateTimePicker1 (dtpDueDate)
10	Label3 (lblSelectedDate)
11	GroupBox2 (gbInvoiceAmount)

Item	Object
12	Label4
13	Label5
14	TextBox1 (txtInvoiceAmount)
15	Button1 (btnProcess)
16	GroupBox3 (gbInvoiceInfo)
17	Label6 (lblPaymentDateDifference)
18	Label7
19	Label8
20	Label9
21	Label10 (lblDiscountRate)
22	Label11 (lblDiscountAmount)
23	Label12 (lblAmountPayable)

A program is needed to meet the following requirements:

Initial specification:

A manufacturer of sawn timber sells to a large number of timber yards and DIY shops. Assume the user is processing payments from these customers. Allow the user to input the date an invoice should have been paid by and the total value of the invoice. Calculate how many days late, if any, the payment has been made. If payment has been made 15 or more days before the due date give a 10% discount, otherwise give a 5% discount if it has been paid on time. Output details about whether payment has been made on time, any discount given and the total amount due.

Overview:

1. The program needs these variables declared at the top of the program:

```
Dim decInvoiceAmountEntered, decInvoiceAmountDue, decDiscountGiven As Decimal
Dim intDiscountRate, intDaysDifference As Integer
Dim dtTodaysDate As Date = Date.Today
Dim intDay As Integer = dtTodaysDate.DayOfWeek
Dim dtInvoiceDate As Date
```

2. Throughout the program use the font of your choice, but be consistent.
3. When the program loads, only the part of the form down to the bottom of gbInvoiceDate should be visible.
4. Upon the Form_Load event, set the background colour of the menustrip to be white, and the foreground to be black.
5. The font of the menu strip should be the same as the font on the form.
6. The menu strip should have two items at the top level: **File** and **Help**.
7. File should contain two items: **New Invoice** and **Close**.
8. **Close** should close the program.
9. **New Invoice** should reset the form to its original condition; the form should show only gbInvoiceDate and the value of the date in dtpDueDate should be set to today's date. All labels and text boxes should be initialised.
10. **Help** should display either a new form or a new label explaining how to use the program. There should be a facility to return to the main form.
11. On the Form_Load event, the correct name of the day should be determined and displayed in lblDate as follows:

```
lblDate.Text = "Today's date: " & _
               WeekdayName(1, , FirstDayOfWeek.System) & ", " & dtTodaysDate
```

12. On the CloseUp event of dtpDueDate:
    - the date value of the DateTimePicker should be stored in dtInvoiceDate, and the form should be enlarged to display gbInvoiceAmount.
    - The date stored in dtInvoiceDate should be displayed in lblSelectedDate.
    - The difference between the selected due date and today's date should be determined by using a DateDiff operation, in the following format:

```
intDaysDifference = DateDiff(DateInterval.Day, dtTodaysDate, dtInvoiceDate)
```

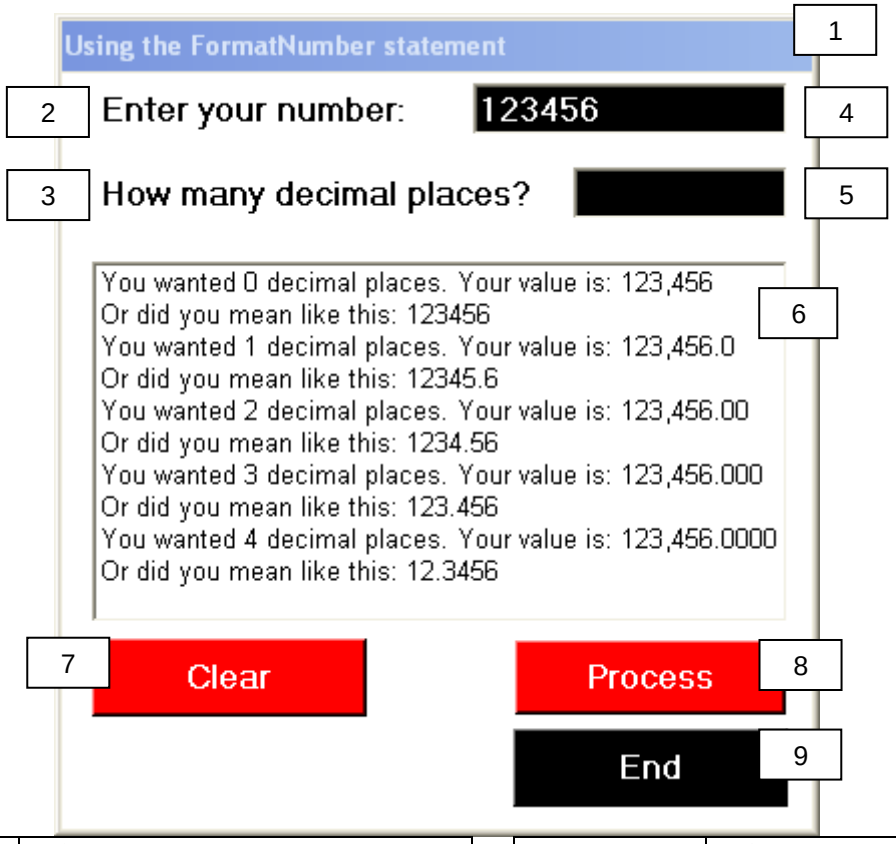
- Focus should be placed in txtInvoiceAmount.

13. On the click event of btnProcess:
    a. Validate the user's input into txtInvoiceAmount as being a number in the range of 1 to 10,000.
    b. Once the number is validated (any non-valid input should be rejected with an appropriate error message and the user returned to the text box for further input), store the value in txtInvoiceAmount into the variable decInvoiceAmountEntered.
    c. Alter the height of the form to display gbInvoiceInfo.
    d. Depending on the value of intDaysDifference, calculate how much discount is to be allocated to the user.
        i. If the value is 14 or more, then 10% discount is given.
        ii. If the value is 0 or more, then 5% discount is given.
        iii. If the value is less than 0, then no discount is given.
    - Store the appropriate discount rate in the variable intDiscountRate (10, 5 or 0).
    - Display an appropriate message to the user in lblPaymentDateDifference (e.g. "You are paying the invoice 15 days early).
    - Display the discount rate being given (intDiscountRate) in lblDiscountRate.
    - Calculate the amount of discount to be given (in monetary terms) by multiplying the value in decInvoiceAmountEntered by intDiscountRate and dividing the whole by 100. Store the answer into the variable decDiscountGiven.
    - Display the value stored in decDiscountGiven (in currency format) in lblDiscountAmount.
    e. Calculate the amount due to be paid by subtracting decDiscountGiven from decInvoiceAmountEntered and storing the answer in decInvoiceAmountDue.
    f. Display the value stored in decInvoiceAmountDue (in currency format) in lblAmountPayable.

Extension activities:

1. Put a splash screen into the project.
2. Put the Help facility onto a separate form.
3. Ensure that the user cannot select a date on the DateTimePicker more than one year in the future or one year in the past.
4. Add ToolTips!

9.0 Specification for program developed in class



Item	Object
1	Form1
2	Label1
3	Label2
4	TextBox1 (txtNumber)
5	TextBox2 (txtDecimalPlaces)

Item	Object
6	ListBox1 (lstOutput)
7	Button1 (btnClear)
8	Button2 (btnProcess)
9	Button3 (btnEnd)

A program is needed to meet the following requirements:

Overview

1. When the program loads, the entire form should be visible.
2. Throughout the program use the font of your choice.
3. Upon form load, focus should be placed in txtNumber.
4. The program needs these variables declared at the top of the program:

```
Dim decNumber As Decimal
Dim intNumberOfDecimalPlaces As Integer
Dim decAnswer As Decimal
```

5. The **Clear** button should clear the text boxes, list box and put the focus back into txtNumber.
6. The **End** button should close the program.
7. **btnProcess** should work with each input as follows:
   a. Input to txtNumber should be validated as being numeric and within the range of 1 to 9,999,999,999 (10 significant digits). Any illegal input should result in an error message in a message box and the cursor placed back in a blank txtNumber.
   b. Valid input in txtNumber should be stored in intNumber.
   c. Input to txtDecimalPlaces should be validated as being numeric and within the range of 0 to 4. Any illegal input should result in an error message in a message box and the cursor placed back in a blank txtDecimalPlaces.
   d. Valid input in txtDecimalPlaces should be stored in intNumberOfDecimalPlaces.
   e. A message to the user should be displayed in lstOutput in the following format:

```
lstOutput.Items.Add("You wanted " & intNumberOfDecimalPlaces & _
      " decimal places. Your value is: " & _
      FormatNumber(decNumber, intNumberOfDecimalPlaces))
```

   f. decAnswer should be calculated using a Select..Case based on the value in intNumberOfDecimalPlaces.
      i. If it is 0, then decAnswer = decNumber.
      ii. If it is 1, then decAnswer = decNumber / 10.
      iii. If it is 2, then decAnswer = decNumber / 100.
      iv. If it is 3, then decAnswer = decNumber / 1000.
      v. If it is 4, then decAnswer = decNumber / 10000.
   g. The output should displayed as a message to the user in lstOutput, in the following format:

```
lstOutput.Items.Add("Or did you mean like this: " & decAnswer)
```

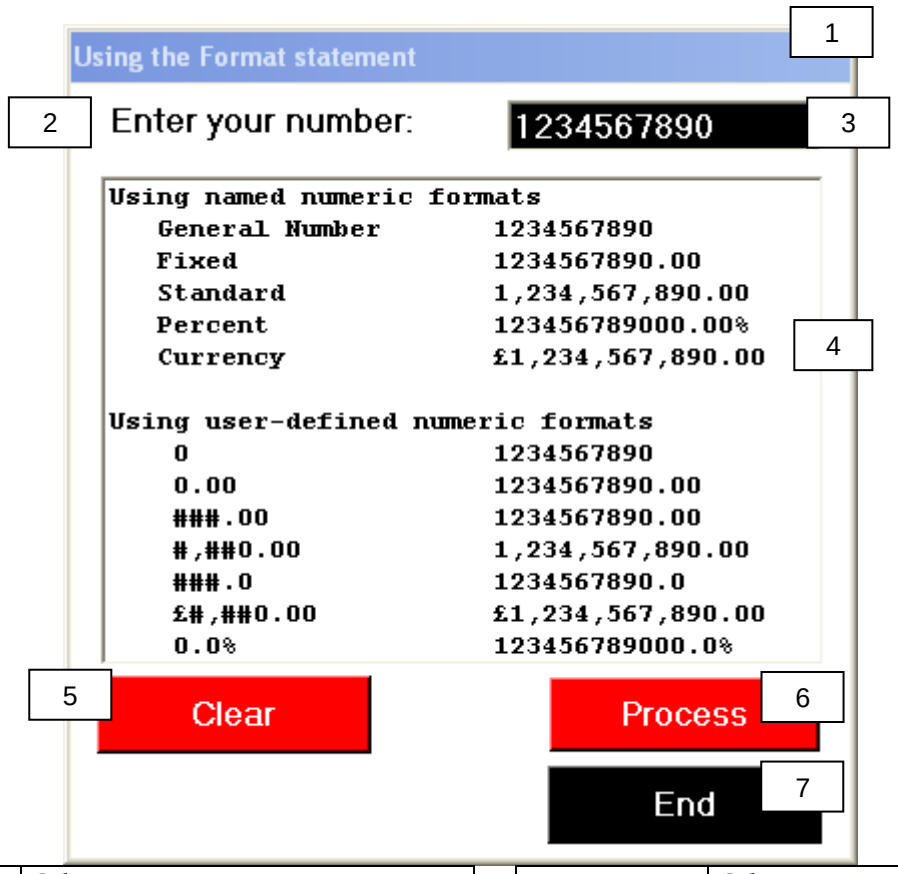
   h. The text should then be cleared from txtDecimalPlaces, and the focus of the cursor put there.

Coding for this program

```
Public Class Form2

    Dim decNumber As Decimal    'the holder of the number
    Dim intNumberOfDecimalPlaces As Integer

    Private Sub btnEnd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnEnd.Click

        'Stop the project
        Me.Close()
    End Sub

    Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClear.Click

        'Clear out the rubbish
        lstOutput.Items.Clear()
        txtNumber.Text = ""
        txtNumber.Focus()
    End Sub

    Private Sub btnProcess_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnProcess.Click

        'Take the value entered in the text box, validate it and display it
        'in different ways.
        If txtNumber.Text.Length = 0 Then
            MsgBox("You must enter something for number!", MsgBoxStyle.Critical, _
              "No entry made for number!")
            txtNumber.Focus()
            Exit Sub
        End If

        If txtNumber.Text.Length > 10 Then
            MsgBox("You have entered too much information in number!", MsgBoxStyle.Critical, _
              "Too long an entry for number!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(Val(txtNumber.Text)) Then
            MsgBox("You must enter a number!", MsgBoxStyle.Critical, "Not numeric!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        decNumber = Val(txtNumber.Text)

        If txtDecimalPlaces.Text.Length = 0 Then
            MsgBox("You must enter something!", MsgBoxStyle.Critical, _
              "No entry made for number of decimal places!")
            txtDecimalPlaces.Focus()
            Exit Sub
        End If
```

```
        If txtDecimalPlaces.Text.Length > 4 Then
            MsgBox("You can have up to four decimal places!", MsgBoxStyle.Critical, _
              "Too much information in number of decimal places!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtDecimalPlaces.Text) Then
            MsgBox("You must enter numbers in number of decimal places!", MsgBoxStyle.Critical, _
               "Numbers only please!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        intNumberOfDecimalPlaces = txtDecimalPlaces.Text

        If (intNumberOfDecimalPlaces < 0) Or (intNumberOfDecimalPlaces > 4) Then
            MsgBox("You can have from 0 to 4 decimal places!", MsgBoxStyle.Critical, _
              "Value out of the accepted range for decimal places!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        lstOutput.Items.Add("You wanted " & intNumberOfDecimalPlaces & _
              " decimal places. Your value is: " & _
                 FormatNumber(decNumber, intNumberOfDecimalPlaces))

        'FormatNumber - the first number is what you want to display, and the second is to
        'how many decimal places.
        'However, it doesn't move the decimal place within the digits of the first number -
        'for that you have to do a little bit more work, as below.

        Dim decAnswer As Decimal

        Select Case intNumberOfDecimalPlaces
            Case 0
                decAnswer = decNumber
            Case 1
                decAnswer = decNumber / 10
            Case 2
                decAnswer = decNumber / 100
            Case 3
                decAnswer = decNumber / 1000
            Case 4
                decAnswer = decNumber / 10000
        End Select

        lstOutput.Items.Add("Or did you mean like this: " & decAnswer)

        txtDecimalPlaces.Text = ""
        txtDecimalPlaces.Focus()
    End Sub

End Class
```

Exercise 9.1



Item	Object
1	Form1
2	Label1
3	TextBox1 (txtNumber)
4	ListBox1 (lstOutput)

Item	Object
5	Button1 (btnClear)
6	Button2 (btnProcess)
7	Button3 (btnEnd)

Example programs – 9. Formatting numbers

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

Overview

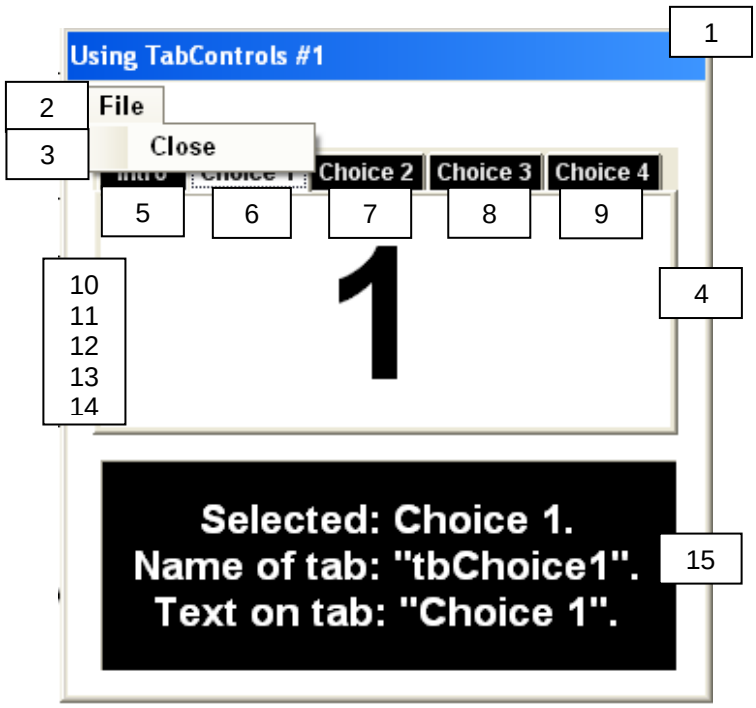
1. When the program loads, the entire form should be visible.
2. Throughout the program use the font of your choice, except for lstOutput which must have a format of Courier New.
3. Upon form load, focus should be placed in txtNumber.
4. The program needs these variables declared at the top of the program:

```
Dim decNumber As Decimal
```

5. The **Clear** button should clear the text box, list box and put the focus back into txtNumber.
6. The **End** button should close the program.
7. **btnProcess** should work with each input as follows:
    a. Input to txtNumber should be validated as being numeric and within the range of 1 to 9,999,999,999 (10 significant digits). Any illegal input should result in an error message in a message box and the cursor placed back in a blank txtNumber.
    b. Valid input in txtNumber should be stored in intNumber.
    c. A message to the user should be displayed in lstOutput in the following format:

```
With lstOutput.Items
     .Add("Using named numeric formats")
     .Add("   General Number       " & Format(decNumber, "General Number"))
     .Add("   Fixed                " & Format(decNumber, "Fixed"))
     .Add("   Standard             " & Format(decNumber, "Standard"))
     .Add("   Percent              " & Format(decNumber, "Percent"))
     .Add("   Currency             " & Format(decNumber, "Currency"))
     .Add(" ")
     .Add("Using user-defined numeric formats")
     .Add("    0                   " & Format(decNumber, "0"))
     .Add("    0.00                " & Format(decNumber, "0.00"))
     .Add("    ###.00              " & Format(decNumber, "###.00"))
     .Add("    #,##0.00            " & Format(decNumber, "#,##0.00"))
     .Add("    ###.0               " & Format(decNumber, "###.0"))
     .Add("    £#,##0.00           " & Format(decNumber, "£#,##0.00"))
     .Add("    0.0%                " & Format(decNumber, "0.0%"))
 End With
```

10.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	CloseToolStripMenuItem(Close)
4	TabControl1 (tbMenu)
5	TabPage1 (tbIntro)
6	TabPage2 (tbChoice1)
7	TabPage3 (tbChoice2)
8	TabPage4 (tbChoice3)
9	TabPage5 (tbChoice4)

Item	Object
10	Label1 (on tbIntro)
11	Label2 (on tbChoice1)
12	Label3 (on tbChoice2)
13	Label4 (on tbChoice3)
14	Label5 (on tbChoice4)
15	Label6 (lblMessage)

Example programs – 10. Using TabControls

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

Overview

The purpose of this program is to illustrate how to make the TabControl object work when the tabs themselves are clicked. The next program (Exercise 10.1) will show how to use them without having to click them.

1. When the program loads, the entire form should be visible.
2. Throughout the program use the font of your choice.
3. The **Close** button should close the program.

4. The TabControl should be renamed to tbMenu.
5. In the TabPages property of tbMenu, five tabs should be declared and named as follows:
    a. First page – tbIntro.
    b. Second – tbChoice1.
    c. Third – tbChoice2.
    d. Fourth – tbChoice3
    e. Fifth – tbChoice4.

6. The TabControl should then be "drawn" using the following code in the DrawItem process of that object:

```
'This subroutine is called every time the tabs are accessed.
'Firstly we'll define some parameters. These are used to "draw" the tabs.
'Tabs aren't objects like we know them - they are "drawn" fresh each time we use
  'them.
Dim CurrentTab As TabPage = tbMenu.TabPages(e.Index)
Dim ItemRect As Rectangle = tbMenu.GetTabRect(e.Index)
Dim FillBrush As New SolidBrush(Color.Black)
Dim TextBrush As New SolidBrush(Color.White)
Dim sf As New StringFormat
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center

'If we are currently painting the Selected TabItem we'll
'change the brush colors and inflate the rectangle.
If CBool(e.State And DrawItemState.Selected) Then
    FillBrush.Color = Color.White
    TextBrush.Color = Color.Black
    ItemRect.Inflate(2, 2)
End If

'Next we'll paint the TabItem with our Fill Brush
e.Graphics.FillRectangle(FillBrush, ItemRect)

'Now draw the text.
e.Graphics.DrawString(CurrentTab.Text, e.Font, TextBrush,
          RectangleF.op_Implicit(ItemRect), sf)

'Reset any Graphics rotation
e.Graphics.ResetTransform()
```

```
'Finally, we should Dispose of our brushes.
FillBrush.Dispose()
TextBrush.Dispose()
```

7. Upon form load, the foreground colour of the menu should match that of the form. The introduction tab should be selected using the SelectedIndex property of tbMenu and setting it to 0. The form load procedure should then call the subroutine "CheckTabs".
8. The CheckTabs subroutine should act as follows:
    a. Using a Select Case on the SelectedIndex of tbMenu:
        i. Case 0 – display "Selected: Introduction" in lblMessage's text property.
        ii. Case 1 – display "Selected: Choice 1" in lblMessage's text property.
        iii. Case 2 – display "Selected: Choice 2" in lblMessage's text
        iv. Case 3 – display "Selected: Choice 3" in lblMessage's text property.
        v. Case 4 – display "Selected: Choice 4" in lblMessage's text property.
    b. Then display the name of the selected tab (i.e. it's tb name) concatenated alongside the current message in the line below on the label using the following code:

```
lblMessage.Text = lblMessage.Text & vbCrLf & _
    "Name of tab: """ & tbMenu.SelectedTab.Name & """."
```

    c. Then in the next line of the label, display the text that is stored within the selected tab, using the following code:

```
lblMessage.Text = lblMessage.Text & vbCrLf & _
    "Text on tab: """ & tbMenu.SelectedTab.Text & """."
```

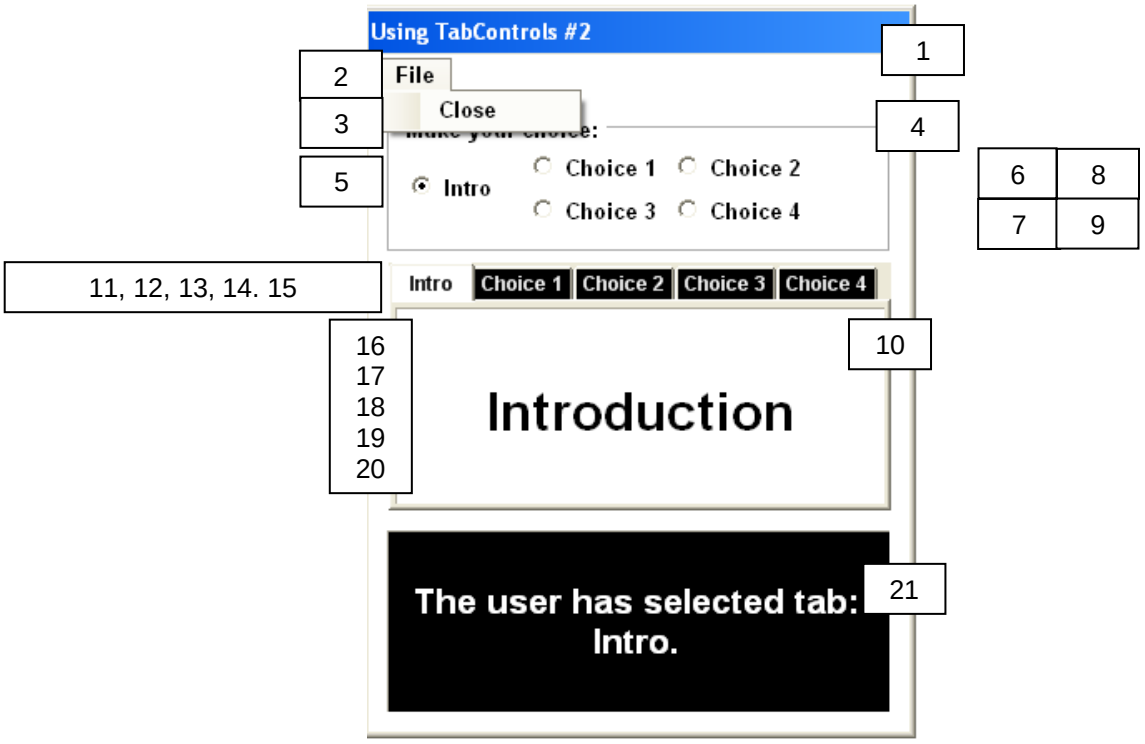
9. From the SelectedIndexChanged event of tbMenu, call the CheckTabs subroutine.

Coding for the program

```
Public Class Form2

    Dim decNumber As Decimal    'the holder of the number
    Dim intNumberOfDecimalPlaces As Integer

    Private Sub btnEnd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnEnd.Click

        'Stop the project
        Me.Close()
    End Sub

    Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnClear.Click

        'Clear out the rubbish
        lstOutput.Items.Clear()
        txtNumber.Text = ""
        txtNumber.Focus()
    End Sub

    Private Sub btnProcess_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnProcess.Click

        'Take the value entered in the text box, validate it and display it
        'in different ways.
        If txtNumber.Text.Length = 0 Then
            MsgBox("You must enter something for number!", MsgBoxStyle.Critical, _
                    "No entry made for number!")
            txtNumber.Focus()
            Exit Sub
        End If

        If txtNumber.Text.Length > 10 Then
            MsgBox("You have entered too much information in number!", MsgBoxStyle.Critical, _
                    "Too long an entry for number!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        If Not IsNumeric(Val(txtNumber.Text)) Then
            MsgBox("You must enter a number!", MsgBoxStyle.Critical, "Not numeric!")
            txtNumber.Text = ""
            txtNumber.Focus()
            Exit Sub
        End If

        decNumber = Val(txtNumber.Text)

        If txtDecimalPlaces.Text.Length = 0 Then
            MsgBox("You must enter something!", MsgBoxStyle.Critical, _
                    "No entry made for number of decimal places!")
            txtDecimalPlaces.Focus()
            Exit Sub
        End If
```

```
        If txtDecimalPlaces.Text.Length > 4 Then
            MsgBox("You can have up to four decimal places!", MsgBoxStyle.Critical, _
                    "Too much information in number of decimal places!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtDecimalPlaces.Text) Then
            MsgBox("You must enter numbers in number of decimal places!", MsgBoxStyle.Critical, _
                    "Numbers only please!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        intNumberOfDecimalPlaces = txtDecimalPlaces.Text

        If (intNumberOfDecimalPlaces < 0) Or (intNumberOfDecimalPlaces > 4) Then
            MsgBox("You can have from 0 to 4 decimal places!", MsgBoxStyle.Critical, _
                    "Value out of the accepted range for decimal places!")
            txtDecimalPlaces.Text = ""
            txtDecimalPlaces.Focus()
            Exit Sub
        End If

        lstOutput.Items.Add("You wanted " & intNumberOfDecimalPlaces & _
              " decimal places. Your value is: " & _
                 FormatNumber(decNumber, intNumberOfDecimalPlaces))

        'FormatNumber - the first number is what you want to display,
        'and the second is to how many decimal places.
        'However, it doesn't move the decimal place within the digits of the first number -
        'for that you have to do
        'a little bit more work, as below.

        Dim decAnswer As Decimal

        Select Case intNumberOfDecimalPlaces
            Case 0
                decAnswer = decNumber
            Case 1
                decAnswer = decNumber / 10
            Case 2
                decAnswer = decNumber / 100
            Case 3
                decAnswer = decNumber / 1000
            Case 4
                decAnswer = decNumber / 10000
        End Select

        lstOutput.Items.Add("Or did you mean like this: " & decAnswer)

        txtDecimalPlaces.Text = ""
        txtDecimalPlaces.Focus()
    End Sub

End Class
```

Exercise 10.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	CloseToolStripMenuItem(Close)
4	GroupBox1
5	RadioButton1 (radIntro)
6	RadioButton2 (radChoice1)
7	RadioButton3 (radChoice2)
8	RadioButton4 (radChoice3)
9	RadioButton5 (radChoice4)
10	TabControl1 (tbMenu)

Item	Object
11	TabPage1 (tbIntro)
12	TabPage2 (tbChoice1)
13	TabPage3 (tbChoice2)
14	TabPage4 (tbChoice3)
15	TabPage5 (tbChoice4)
16	Label1 (on tbChoice1)
17	Label2 (on tbChoice1)
18	Label3 (on tbChoice2)
19	Label4 (on tbChoice3)
20	Label5 (on tbChoice4)
21	Label6 (lblMessage)

Example programs – 10. Using TabControls

Lecturer: Jane Fletcher

A program is needed to meet the following requirements:

Overview

The purpose of this program is to illustrate how to make the TabControl object work without actually clicking on the tabs themselves. It is the follow-up program to the one on the previous pages.

1. When the program loads, the entire form should be visible.

2. Throughout the program use the font of your choice.

3. The **Close** button should close the program.

4. The TabControl should be renamed to tbMenu.

5. In the TabPages property of tbMenu, five tabs should be declared and named as follows:
    a. First page – tbIntro.
    b. Second – tbChoice1.
    c. Third – tbChoice2.
    d. Fourth – tbChoice3
    e. Fifth – tbChoice4.

6. The TabControl should then be "drawn" using the following code in the DrawItem process of that object:

```
'This subroutine is called every time the tabs are accessed.
'Firstly we'll define some parameters. These are used to "draw" the tabs.
'Tabs aren't objects like we know them - they are "drawn" fresh each time we use
  'them.
Dim CurrentTab As TabPage = tbMenu.TabPages(e.Index)
Dim ItemRect As Rectangle = tbMenu.GetTabRect(e.Index)
Dim FillBrush As New SolidBrush(Color.Black)
Dim TextBrush As New SolidBrush(Color.White)
Dim sf As New StringFormat
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center

'If we are currently painting the Selected TabItem we'll
'change the brush colors and inflate the rectangle.
If CBool(e.State And DrawItemState.Selected) Then
    FillBrush.Color = Color.White
    TextBrush.Color = Color.Black
    ItemRect.Inflate(2, 2)
End If

'Next we'll paint the TabItem with our Fill Brush
e.Graphics.FillRectangle(FillBrush, ItemRect)

'Now draw the text.
e.Graphics.DrawString(CurrentTab.Text, e.Font, TextBrush, _
  RectangleF.op_Implicit(ItemRect), sf)
```

```
'Reset any Graphics rotation
e.Graphics.ResetTransform()

'Finally, we should Dispose of our brushes.
FillBrush.Dispose()
TextBrush.Dispose()
```

7. Upon form load, the foreground colour of the menu should match that of the form.

8. For each of the radio buttons' CheckedChange event:
    a. If the Checked property of the radio button is "True":
        i. Set the SelectedIndex property of tbMenu to be equal to:
            1. For tbIntro, 0
            2. For tbChoice1, 1
            3. For tbChoice2, 2
            4. For tbChoice3, 3
            5. For tbChoice4, 4

        ii. display a message in lblMessage to display a message saying "The user has selected tab: " followed by the text of the selected tab. The code for this is:

```
lblMessage.Text = "The user has selected tab: " & _
            tbMenu.SelectedTab.Text & "."
```

    b. If the Checked property is false, set the text in lblMessage to be blank.

11.0 Specification for program developed in class

Item	Object
1	Form1
2	MenuStrip1 (File)
3	BackToolStripMenuItem(Back)
4	ForwardToolStripMenuItem(Forward)
5	ExitToolStripMenuItem(Exit)
6	Label1

Item	Object
7	TextBox1 (txtAddress)
8	Button1 (btnGo)
9	WebBrowser1(wbWebBrowser)

A program is needed to meet the following requirements:

Overview

The purpose of this program is to create the very simplest of web browsers.

1. In design mode, set the Url property of wbWebBrowser to www.snc.ac.uk.

2. When the program loads, the entire form should be visible. The web browser itself should stretch across the entire screen, regardless of the size of that screen. The code for this, which should be placed in the form load event is:

```
Dim ScreenSize As Size = New Size(Screen.PrimaryScreen.Bounds.Width, _
    Screen.PrimaryScreen.Bounds.Height)
Dim intWidth, intHeight As Integer

intWidth = Screen.PrimaryScreen.Bounds.Width
intHeight = Screen.PrimaryScreen.Bounds.Height

wbWebBrowser.Left = 20
wbWebBrowser.Width = intWidth - 40
wbWebBrowser.Height = intHeight - 150
```

3. Throughout the program use the font of your choice.

4. The **Exit** button should close the program.

5. The **Forward** button should move the browser on to the last website accessed (should the **Back** button have been pressed). You do not have to check for a history.

```
wbWebBrowser.GoForward()
```

6. The **Back** button should move the browser to the previous website accessed. You do not have to check for a history.

```
wbWebBrowser.GoBack()
```

7. The processing for btnGo is as follows:
   i. Firstly ensure that the user has entered something by checking the length property of the text box txtAddress. If the length is 0, then the subrouting should be exited with no further processing.
   j. Declare a string variable called strAddress.
   k. Move the contents of text property of txtAddress to strAddress.
   l. Change the contents of the variable strAddress to lower case:

```
strAddress = LCase(strAddress)
```

   m. Declare a string variable called strFrontBit.

n. Determine the first four characters in the variable strAddress to ensure that the user has entered “www.”.

```
strFrontBit = Mid(strAddress, 1, 4)
```

o. Look at the contents of strFrontBit. If it is NOT equal to “www.” then concatenate “www.” onto the front of strAddress.

```
If strFrontBit = "www." Then
Else
    strAddress = "www." & strAddress
End If
```

p. Navigate to the webpage indicated in strAddress:

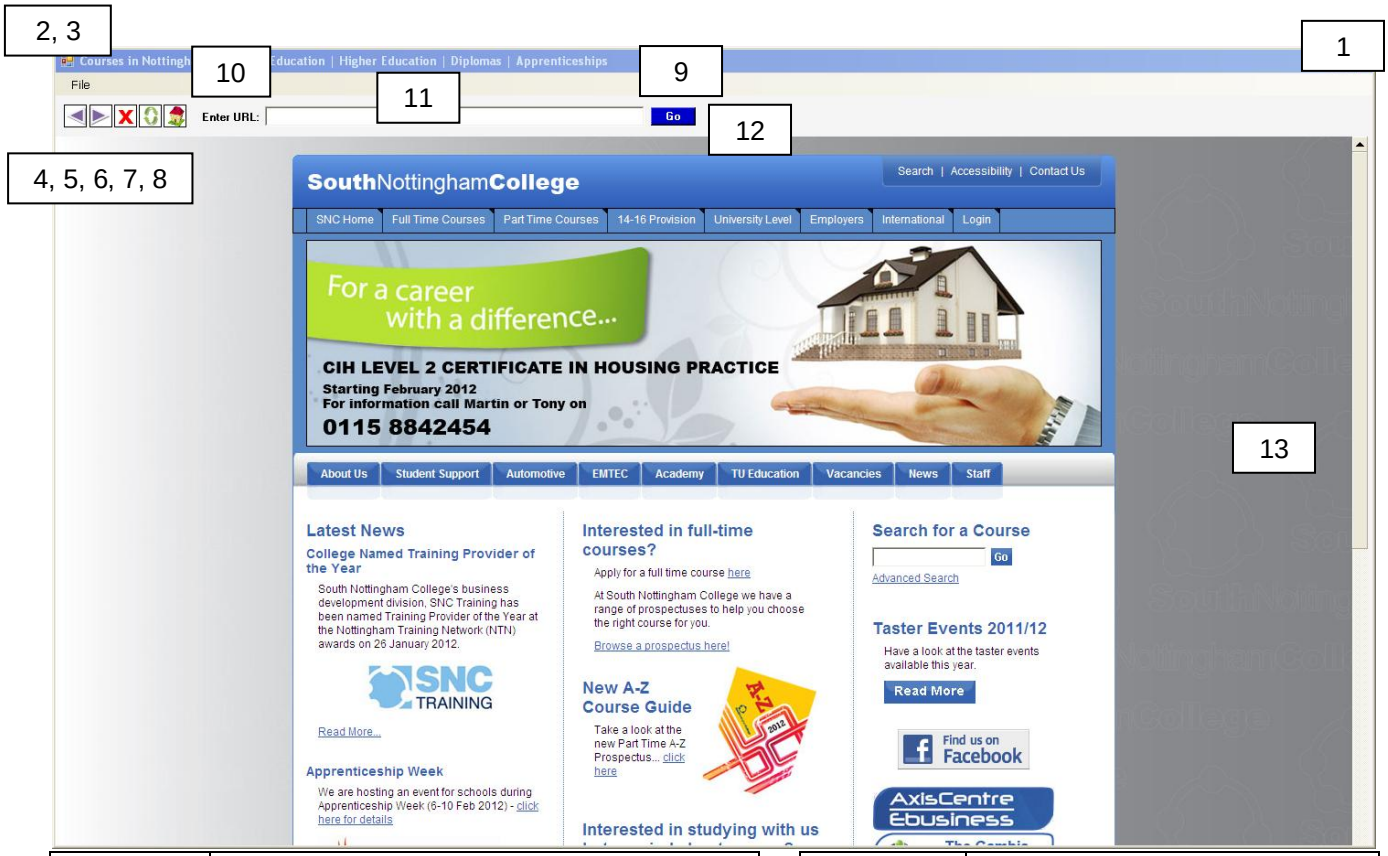
```
wbWebBrowser.Navigate(strAddress)
```

Code for this program

```
Public Class Form1


    Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnGo.Click

        'This button will check that the user has entered something and then will navigate
        'to the web address.

        Dim strAddress As String

        If txtAddress.Text.Length = 0 Then
            Exit Sub
        End If

        strAddress = txtAddress.Text

        strAddress = LCase(strAddress)

        Dim strFrontBit As String

        strFrontBit = Mid(strAddress, 1, 4)

        If strFrontBit = "www." Then
        Else
            strAddress = "www." & strAddress
        End If

        wbWebBrowser.Navigate(strAddress)
    End Sub


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        Dim ScreenSize As Size = New Size(Screen.PrimaryScreen.Bounds.Width, _
              Screen.PrimaryScreen.Bounds.Height)
        Dim intWidth, intHeight As Integer

        intWidth = Screen.PrimaryScreen.Bounds.Width
        intHeight = Screen.PrimaryScreen.Bounds.Height

        wbWebBrowser.Left = 20
        wbWebBrowser.Width = intWidth - 40
        wbWebBrowser.Height = intHeight - 150

    End Sub

    Private Sub BackToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles BackToolStripMenuItem.Click

        wbWebBrowser.GoBack()

    End Sub
```

```
    Private Sub ForwardToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles ForwardToolStripMenuItem.Click

        wbWebBrowser.GoForward()

    End Sub

    Private Sub ExitToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
              Handles ExitToolStripMenuItem.Click

        End
    End Sub
End Class
```

Exercise 11.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	ExitToolStripMenuItem(Exit)
4	Button1(btnBack)
5	Button2 (btnForward)
6	Button3 (btnStop)
7	Button4 (btnRefresh)

Item	Object
8	Button5 (btnHome)
9	GroupBox1
10	Label1
11	TextBox1 (txtWebAddress)
12	Button6 (btnGo)
13	WebBrowser1 (wbWebBrowser)

A program is needed to meet the following requirements:

Overview

The purpose of this program is to create a more complex web browser. This program also introduces ToolTips.

1. In design mode, set the Url property of wbWebBrowser to www.snc.ac.uk.

2. Add a ToolTips control from the ToolBox in design mode. Name it ttWebBrowserBits.

3. When the program loads, the entire form should be visible. The web browser itself should stretch across the entire screen, regardless of the size of that screen. The code for this, which should be placed in the form load event is:

```
Dim ScreenSize As Size = New Size(Screen.PrimaryScreen.Bounds.Width, _
    Screen.PrimaryScreen.Bounds.Height)
Dim intWidth, intHeight As Integer

intWidth = Screen.PrimaryScreen.Bounds.Width
intHeight = Screen.PrimaryScreen.Bounds.Height

wbWebBrowser.Left = 20
wbWebBrowser.Width = intWidth - 40
wbWebBrowser.Height = intHeight - 150
```

4. Also in form load, allocate the tool tips to the objects on the form that require them.

```
Me.ttWebBrowserBits.ShowAlways = True
Me.ttWebBrowserBits.ReshowDelay = 10
Me.ttWebBrowserBits.AutomaticDelay = 100

ttWebBrowserBits.SetToolTip(btnBack, "Back")
ttWebBrowserBits.SetToolTip(btnForward, "Forward")
ttWebBrowserBits.SetToolTip(btnStop, "Stop")
ttWebBrowserBits.SetToolTip(btnRefresh, "Refresh")
ttWebBrowserBits.SetToolTip(btnHome, "Home")
ttWebBrowserBits.SetToolTip(txtWebAddress, "Enter Web Address")
```

5. Throughout the program use the font of your choice.

6. The **Exit** button should close the program.

7. The **Forward** button should move the browser on to the last website accessed (should the **Back** button have been pressed). You do not have to check for a history.

```
wbWebBrowser.GoForward()
```

8. The **Back** button should move the browser to the previous website accessed. You do not have to check for a history.

```
wbWebBrowser.GoBack()
```

9. The **Home** button should move the browser to the "home url" of www.snc.ac.uk.

```
wbWebBrowser.Navigate("www.snc.ac.uk")
```

10. The processing for btnGo is as follows:
    q. Firstly ensure that the user has entered something by checking the length property of the text box txtWebAddress. If the length is 0, then the subroutine should show an error message via a message box and exit with no further processing.
    r. Declare a string variable called strWebAddress.
    s. Move the contents of text property of txtWebAddress to strWebAddress.
    t. Change the contents of the variable strWebAddress to lower case:

```
strWebAddress = LCase(strWebAddress)
```

    u. Declare a string variable called strPrefix.
    v. Determine the first four characters in the variable strWebAddress to ensure that the user has entered "www.".

```
strPrefix = Mid(strWebAddress, 1, 4)
```

    w. Look at the contents of strPrefix. If it is NOT equal to "www." then concatenate "www." onto the front of strWebAddress.

```
If strPrefix = "www." Then
Else
    strWebAddress = "www." & strWebAddress
End If
```

    x. Navigate to the webpage indicated in strWebAddress:

```
wbWebBrowser.Navigate(strWebAddress)
```

11. The "Navigated" event of the web browser should display the title of the web document being displayed in the Text property of the form.

```
Me.Text = wbWebBrowser.DocumentTitle.ToString
```

12. The "Navigating" event of the web browser should display the name of the website being found in the Text property of the form.

```
Me.Text = "Finding..." & strWebAddress
```

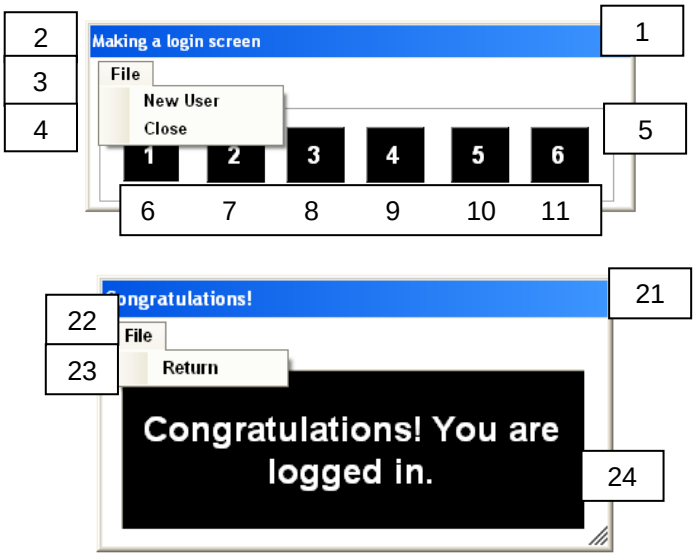
Extension activities

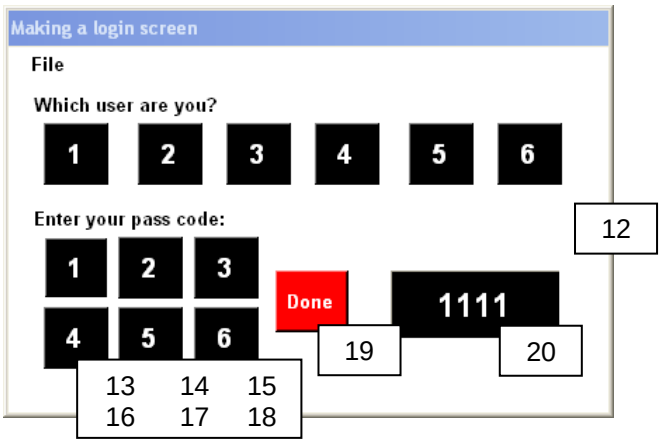
1. Add a combo box to search using Google:

```
wbWebBrowser.Navigate("http://www.google.co.uk/search?hl=en&q=" + _
        txtSearch.Text)
```

2. Change the AcceptButton of the form to be context-specific, as per the example program called “WebBrowserWithExtensionActivities” (found on the S drive in folder Week 20 for this unit).

12.0 Specification for program developed in class





Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewUserToolStripMenuItem(New User)
4	CloseToolStripMenuItem(Close)
5	GroupBox1
6	Button1 (btnUser1)
7	Button2 (btnUser2)
8	Button3 (btnUser3)
9	Button4 (btnUser4)
10	Button5 (btnUser5)
11	Button6 (btnUser6)
12	GroupBox2

Item	Object
13	Button7 (btnPass1)
14	Button8 (btnPass2)
15	Button9 (btnPass3)
16	Button10 (btnPass4)
17	Button11 (btnPass5)
18	Button12 (btnPass6)
19	btnDone
20	lblPassCode
21	Form2
22	MenuStrip1 (File)
23	ReturnToolStripMenuItem(Return)
24	Label1

A program is needed to meet the following requirements:

Overview

The purpose of this program is to create the very simplest login screen.

1. When the program loads, only the top part the form should be visible.

2. Throughout the program use the font of your choice.

3. Declare two variables at the top of the program:

```
Dim intUser As Integer
Dim strPassword As String
```

4. The **Close** button should close the program.

5. The **New User** button should initialise strPassword to a null value, set lblPassCode's text to blank and reset the form to its original size.

6. Once the user has clicked on one of the numbered buttons to indicate which user they are, the screen should be enlarged to display the second group box. strPassword should be set to equal null value and intUser should be assigned the value of the sender's text. NOTE: the "User" buttons will all work under the same subroutine – the programmer should add each button's click event to the procedure for btnUser1_Click, as shown below:

```
Private Sub btnUser1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnUser1.Click, btnUser2.Click, _
    btnUser3.Click, btnUser4.Click, btnUser5.Click, _
    btnUser6.Click
```

7. The buttons btnPass1 – btnPass6 will work in a similar fashion as in step 6. For each click of a button, the variable strPassword should be concatenated with the sender's text:

```
strPassword = strPassword & sender.text
```

   and the value currently in strPassword should be displayed in lblPassCode.

8. When the user clicks btnDone, the program should examine the value in intUser and check to see if the correct passcode for that user has been entered, using a Select..Case statement. For user 1, the passcode is 1111; for user 2 it is 2222 and so on.
   a. Declare a Boolean variable called bValid.
   b. Set the value of bValid to be false.
   c. For each case of intUser (1 – 6), if the passcode entered is correct, set bValid to true.

9. If the user enters a valid passcode (i.e. bValid is True), then the program should display Form2 and wait for the user to click that form's "Return" button. If an invalid passcode has been entered then the program must display an error message in the form of a message box.

10. The "Return" menu item on Form2 should dispose of Form2 and return control back to Form1.

Coding for this program

FORM1

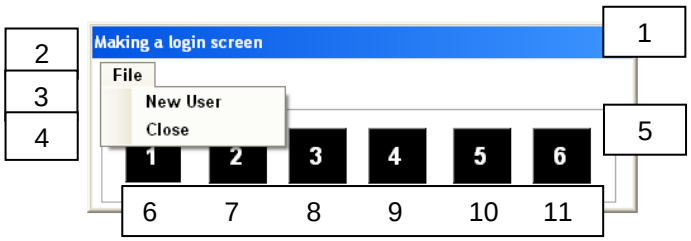
```
Public Class Form1

    Dim intUser As Integer
    Dim strPassword As String

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        'Set the defaults of the form.
        Me.Height = 151
        MenuStrip1.BackColor = Color.White
    End Sub

    Private Sub btnUser1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnUser1.Click, btnUser2.Click, _
        btnUser3.Click, btnUser4.Click, btnUser5.Click, btnUser6.Click

        'Get the identification of the user logging into the system.
        intUser = sender.text
        strPassword = "" 'Blank out previous input to the password.
        Me.Height = 294
    End Sub

    Private Sub btnPass1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnPass1.Click, btnPass2.Click, btnPass3.Click,
                btnPass4.Click, btnPass5.Click, btnPass6.Click

        'Concatenate the text on the clicked button with previous clicks
        'to form the passcode of the user.
        strPassword = strPassword & sender.text
        lblPassCode.Text = strPassword

    End Sub

    Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click

        'End the project
        Me.Dispose()
        End
    End Sub

    Private Sub btnDone_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnDone.Click

        'The user has entered his or her own passcode so check to see if
        'it is valid or not.

        Dim bValid As Boolean
        bValid = False

        Select Case intUser
            Case 1
                If strPassword = "1111" Then
```

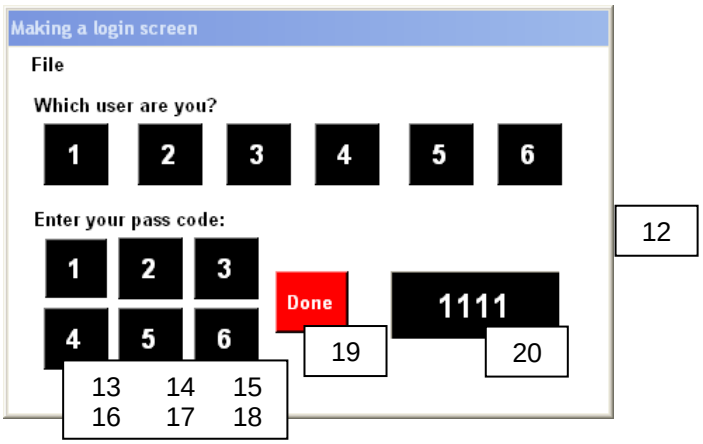
```
                    bValid = True
                End If
            Case 2
                If strPassword = "2222" Then
                    bValid = True
                End If
            Case 3
                If strPassword = "3333" Then
                    bValid = True
                End If
            Case 4
                If strPassword = "4444" Then
                    bValid = True
                End If
            Case 5
                If strPassword = "5555" Then
                    bValid = True
                End If
            Case 6
                If strPassword = "6666" Then
                    bValid = True
                End If
        End Select

        If bValid Then
            Dim form2 As Form2 = New Form2
            form2.ShowDialog()
        Else
            MsgBox("You have entered an incorrect passcode.", MsgBoxStyle.Critical, "ERROR!")
        End If
    End Sub

    Private Sub NewUserToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles NewUserToolStripMenuItem.Click

        'Clear everything down and reset it.
        strPassword = ""
        lblPassCode.Text = ""
        Me.Height = 151
    End Sub
End Class
```

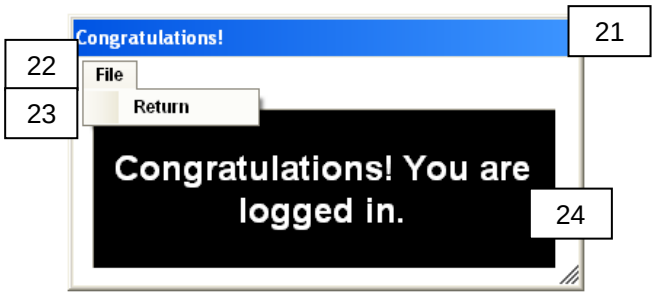
FORM2

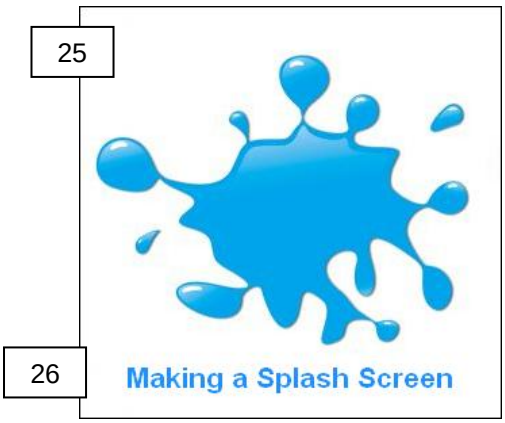
```
Public Class Form2

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        MenuStrip1.BackColor = Color.White
    End Sub

    Private Sub ReturnToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles ReturnToolStripMenuItem.Click

        'Get rid of this form and return.
        Me.Dispose()
        Return
    End Sub
End Class
```

Exercise 12.1









Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewUserToolStripMenuItem(New User)
4	CloseToolStripMenuItem(Close)
5	GroupBox1
6	Button1 (btnUser1)
7	Button2 (btnUser2)
8	Button3 (btnUser3)
9	Button4 (btnUser4)
10	Button5 (btnUser5)
11	Button6 (btnUser6)
12	GroupBox2
13	Button7 (btnPass1)

Item	Object
14	Button8 (btnPass2)
15	Button9 (btnPass3)
16	Button10 (btnPass4)
17	Button11 (btnPass5)
18	Button12 (btnPass6)
19	btnDone
20	lblPassCode
21	Form2
22	MenuStrip1 (File)
23	ReturnToolStripMenuItem(Return)
24	Label1
25	SplashScreen1
26	Label1

A program is needed to meet the following requirements:

Overview

The purpose of this program is to create the very simplest login screen, but this time with the inclusion of a splash screen as well.

1. When the program loads, the splash screen should be the start-up screen. Go to the Properties menu of the project, then select the last item in that drop-down menu (the properties of your project) and pick the "Application" tab. Make sure that your start-up form is set to the splash screen. Set the background image of the form to the image of your choice, with the proviso that it is an appropriate image and will in no way be offensive to any other person.

2. The splash screen should have a timer with an interval set to 1000. The timer should be set to disabled at design time.

3. The form load event of the splash screen should enable the timer.

4. The timer's tick event (the default event for a timer) should disable the timer, then declare your first form – probably Form1. It should then hide itself and show dialog the recently declared form:

```
Dim Form1 As Form1 = New Form1
Me.Hide()
Form1.ShowDialog()
```

5. When Form1 loads, only the top part the form should be visible.

6. Throughout the program use the font of your choice.

7. Declare two variables at the top of the program:

```
Dim intUser As Integer
Dim strPassword As String
```

8. The **Close** button should close the program.

9. The **New User** button should initialise strPassword to a null value, set lblPassCode's text to blank and reset the form to its original size.

10. Once the user has clicked on one of the numbered buttons to indicate which user they are, the screen should be enlarged to display the second group box. strPassword should be set to equal null value and intUser should be assigned the value of the sender's text. NOTE: the "User" buttons will all work under the same subroutine – the programmer should add each button's click event to the procedure for btnUser1_Click, as shown below:

```
Private Sub btnUser1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) _
      Handles btnUser1.Click, btnUser2.Click, _
      btnUser3.Click, btnUser4.Click, btnUser5.Click, _
      btnUser6.Click
```

11. The buttons btnPass1 – btnPass6 will work in a similar fashion as in step 6. For each click of a button, the variable strPassword should be concatenated with the sender's text:

```
strPassword = strPassword & sender.text
```

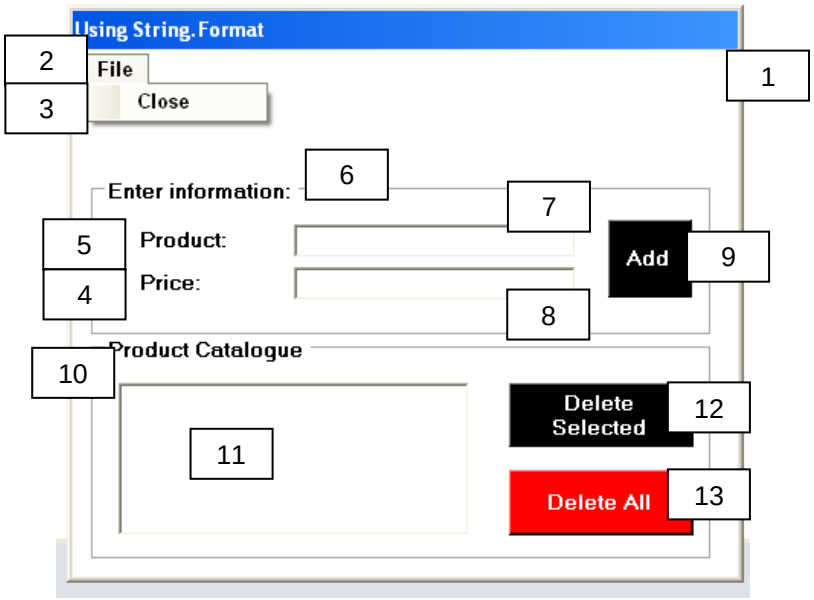
and the value currently in strPassword should be displayed in lblPassCode.

12. When the user clicks btnDone, the program should examine the value in intUser and check to see if the correct passcode for that user has been entered, using a Select..Case statement. For user 1, the passcode is 1111; for user 2 it is 2222 and so on.
    a. Declare a Boolean variable called bValid.
    b. Set the value of bValid to be false.
    c. For each case of intUser (1 – 6), if the passcode entered is correct, set bValid to true.

13. If the user enters a valid passcode (i.e. bValid is True), then the program should display Form2 and wait for the user to click that form's "Return" button. If an invalid passcode has been entered then the program must display an error message in the form of a message box.

14. The "Return" menu item on Form2 should dispose of Form2 and return control back to Form1.

13.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	CloseToolStripMenuItem(Close)
4	Label1
5	Label2
6	GroupBox1
7	TextBox1 (txtProduct)

Item	Object
8	TextBox2 (txtPrice)
9	Button1 (btnAdd)
10	GroupBox2
11	ListBox1 (lstCatalogue)
12	Button2 (btnDeleteSelected)
13	Button3 (btnDeleteAll)

A program is needed to meet the following requirements:

Overview

The purpose of this program is to refresh your memory on using listboxes.

1. Throughout the program use the font of your choice.

2. When the form loads, ensure that the background colour of the menustrip is the same as the background colour of the form. Place focus in the "Product" text box.

3. Declare one variable at the top of the program:

```
Dim MyFormat As String = "{0, -20} {1, -15}"
```

4. The **Close** button should close the program.

5. Coding for btnAdd:
   a. btnAdd should validate user input – the text box for product should contain up to 20 characters, and must exist. The text in price should be numeric, and in the range of 0.01 to 999.999. Any erroneous input should result in a message box to the user and the cursor replaced into the correct text box to await further user entry.

   b. Declare a local decimal variable for the contents of the price textbox, called decPrice. The validated input from txtPrice should be stored within that once validated.

   c. Once input is valid, it should be displayed in the list box using the following statement:

```
lstCatalogue.Items.Add(String.Format(MyFormat, txtProduct.Text, _
    Format(decPrice, "Currency")))
```

   d. Both text boxes should be cleared of input and the cursor placed in txtProduct to await user input.

6. btnDeleteSelected should check to see that the user has selected any line within the list box by examining the list box's SelectedIndex property. If nothing has been selected then a message box should be displayed to the user telling them that they must pick a line to delete. If a line has been selected, then it should be deleted and the cursor put back into txtProduct to await user input.

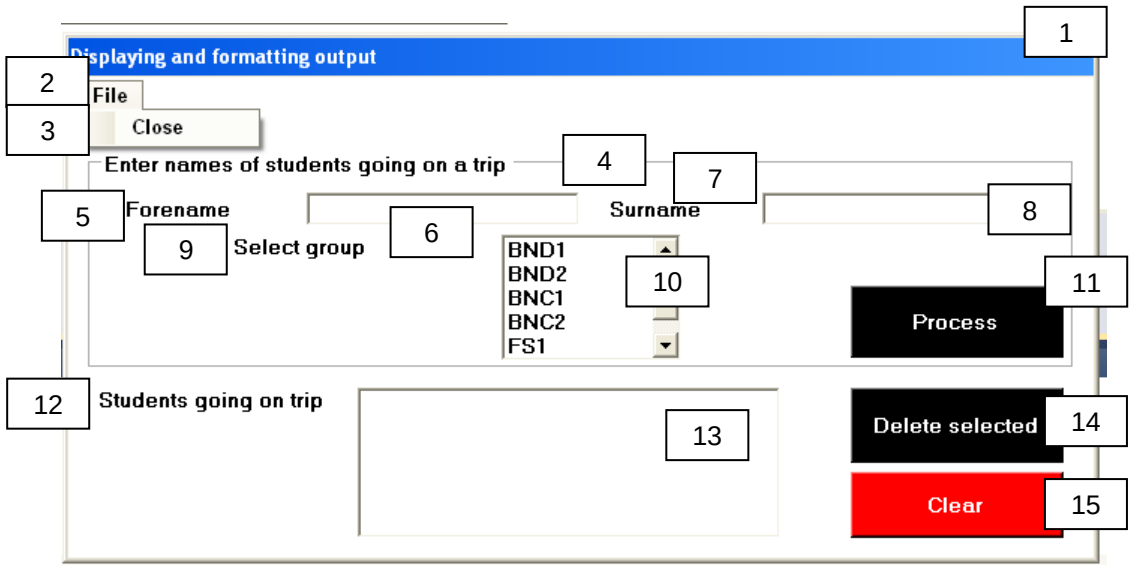
7. btnDeleteAll should first check to see that there are some items in the list box by examining the list box's Count property. If there is nothing in the list box then the user should be given an appropriate error message via a list box. Should there be data in the list box then the user should be asked via a message box if they wish to delete all their catalogue of data. If the response is yes, then the list box should be cleared of all input. In either case, the cursor should be returned to txtProduct to await user input.

Coding for this program

```
Public Class Form1

    Dim MyFormat As String = "{0, -20} {1, -15}"

    Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click

        'End the project
        Me.Dispose()
        End
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles MyBase.Load

        'Set up the defaults of the form
        MenuStrip1.BackColor = Color.White

    End Sub

    Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
              Handles btnAdd.Click

        'Perform the usual validation to make sure that user entry is correct.
        If txtProduct.Text.Length = 0 Then
            MsgBox("You must enter a product before continuing!", MsgBoxStyle.Critical, "ERROR!")
            txtProduct.Focus()
            Exit Sub
        End If

        If txtProduct.Text.Length > 20 Then
            MsgBox("You may enter up to 20 characters for product name.", MsgBoxStyle.Critical, _
                      "ERROR!")
            txtProduct.Text = ""
            txtProduct.Focus()
            Exit Sub
        End If

        If txtPrice.Text.Length = 0 Then
            MsgBox("You must enter a price before continuing!", MsgBoxStyle.Critical, "ERROR!")
            txtPrice.Focus()
            Exit Sub
        End If

        If txtPrice.Text.Length > 5 Then
            MsgBox("You must enter a price in the range of £1 to £999.99!", MsgBoxStyle.Critical, _
                      "ERROR!")
            txtPrice.Text = ""
            txtPrice.Focus()
            Exit Sub
        End If

        If Not IsNumeric(txtPrice.Text) Then
            MsgBox("You must enter a price in the range of £1 to £999.99!", MsgBoxStyle.Critical, _
                      "ERROR!")
```

```
            txtPrice.Text = ""
            txtPrice.Focus()
            Exit Sub
        End If

        Dim decPrice As Decimal
        decPrice = txtPrice.Text

        If (decPrice < 0.01) Or (decPrice > 999.99) Then
            MsgBox("You must enter a price in the range of £1 to £999.99!", MsgBoxStyle.Critical, _
                   "ERROR!")
            txtPrice.Text = ""
            txtPrice.Focus()
            Exit Sub
        End If

        lstCatalogue.Items.Add(String.Format(MyFormat, txtProduct.Text, Format(decPrice, _
                    "Currency")))
        txtProduct.Text = ""
        txtPrice.Text = ""
        txtProduct.Focus()
    End Sub

    Private Sub btnDeleteSelected_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnDeleteSelected.Click

        'Check to make sure that the user has selected something from the list box.
        If lstCatalogue.SelectedIndex = -1 Then
            MsgBox("You must select something before deleting it!", MsgBoxStyle.Critical, "ERROR!")
            Exit Sub
        End If

        'The user has made a valid selection, so delete it.
        lstCatalogue.Items.RemoveAt(lstCatalogue.SelectedIndex)
        'Put the focus back in the text box.
        txtProduct.Focus()
    End Sub

    Private Sub btnDeleteAll_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnDeleteAll.Click

        'Check to see if there is anything in the list box to delete first.
        If lstCatalogue.Items.Count = 0 Then
            MsgBox("You must have something to delete first!", MsgBoxStyle.Critical, "ERROR!")
            Exit Sub
        End If

        Dim intResponse As Integer
        intResponse = MsgBox("Are you sure you want to delete your entire product catalogue?", _
              MsgBoxStyle.YesNo, "ARE YOU SURE?")

        If intResponse = vbYes Then
            'The user wants to clear all the items from the catalogue
            lstCatalogue.Items.Clear()
        End If

        txtProduct.Focus()
    End Sub
End Class
```

Exercise 13.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	CloseToolStripMenuItem(Close)
4	GroupBox1
5	Label1
6	TextBox1 (txtForename)
7	Label2
8	TextBox2 (txtSurname)

Item	Object
9	Label3
10	ListBox1 (lstGroup)
11	Button1 (btnProcess)
12	Label4
13	ListBox2 (lstStudentsOnTrip)
14	Button2 (btnDeleteSelected)
15	Button3 (btnClear)

A program is needed to meet the following requirements:

Overview

The purpose of this program is to refresh your memory on using listboxes.

1. Throughout the program use the font of your choice.

2. When the form loads, ensure that the background colour of the menustrip is the same as the background colour of the form. Place focus in the "Forename" text box.

3. Declare these variables at the top of the program:

```
Dim MyFormat As String = "{0, -10} {1, -15} {2, -5}"

Dim strForename As String
Dim strSurname As String
Dim strGroup As String
```

4. The **Close** button should close the program.

5. Coding for btnProcess:
   a. btnProcess should validate user input – the input for each item should exist and if not this should result in a message box to the user and the cursor replaced into the correct text box to await further user entry. If input exists, then it should be moved to either strForename or strSurname and converted to proper case (i.e. a capital letter for the first letter of the word):

```
strForename = StrConv(txtForename.Text, VbStrConv.ProperCase)
```

   b. Once this has been accomplished, btnProcess should check to ensure that the user has selected a group from lstGroup. If not, an error message should be displayed via a message box. If a group has been selected, then it should be stored in strGroup and the SelectedIndex property of lstGroup should be set to -1.

   c. Once input is valid, it should be displayed in the list box using the following statement:

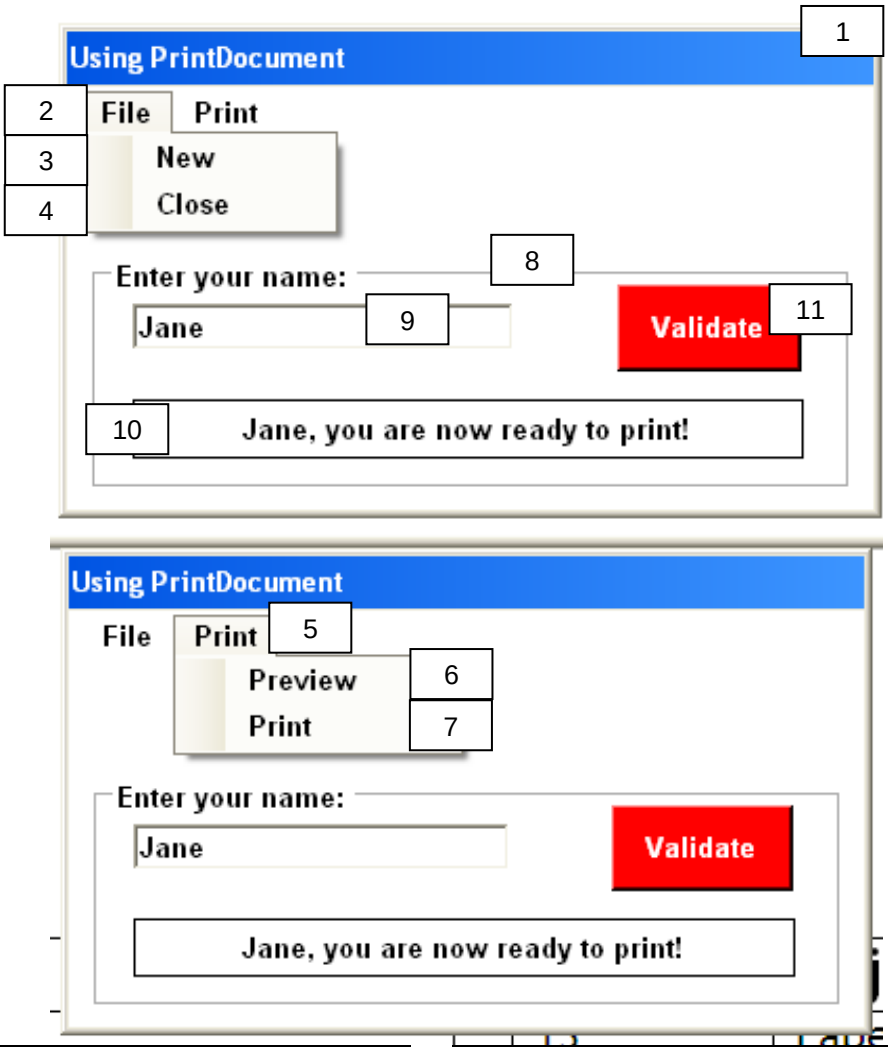
```
lstStudentsOnTrip.Items.Add(String.Format(MyFormat, _
            strForename, strSurname, strGroup))
```

   d. Both text boxes should be cleared of input and the cursor placed in txtForename to await user input.

6. btnDeleteSelected should check to see that the user has selected any line within the list box by examining the list box's SelectedIndex property. If nothing has been selected then a message box should be displayed to the user telling them that they must pick a line to delete. If a line has been selected, then it should be deleted and the cursor put back into txtForename to await user input.

7. btnClear should first check to see that there are some items in the list box by examining the list box's Count property. If there is nothing in the list box then the user should be given an appropriate error message via a list box. Should there be data in the list box then the user should be asked via a message box if they wish to delete all their catalogue of data. If the response is yes, then the list box should be cleared of all input. In either case, the cursor should be returned to txtForename to await user input.

14.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	MenuStrip1 (Print)
6	PreviewToolStripMenuItem

Item	Object
7	PrintToolStripMenuItem
8	GroupBox1
9	TextBox1 (txtName)
10	Label1 (lblStatus)
11	Button1 (btnValidate)
12	(Hidden) PrintPreviewDialog
13	(Hidden) PrintDocument1

A program is required to meet the following requirements:

This program introduces the concept of printing within a .NET environment.

1. The menu strip should have two items at the top level: File and Print.
2. File should contain two items: New and Close.
3. Close should close the program.
4. New should reset the form to its original condition and all variables should be set to their original values. The Print option on the menu strip should be disabled.
5. When the program loads, everything should be visible on the form and the background colour of the menu strip set to whatever is used within the program for the background colour of the form. The Print option on the menu strip should be disabled.
6. The program needs this variable to be declared at the top of the program: strName as string.
7. At design time, PrintDocument1 should be assigned as the "Document" in the PrintPreviewDialogue control.
8. The processing for btnValidate should be as follows:
    a. The user's entry into txtName should be validated to be within the length of 1 to 20 characters. Anything outside that range should be rejected with an appropriate error message. The user should be exited from that procedure.
    b. Valid input should be stored in strName.
    c. The message "*name*, you are now ready to print!" should be displayed in lblStatus, where *name* is the value stored in strName.
9. The Preview menu strip click event should contain the following code:

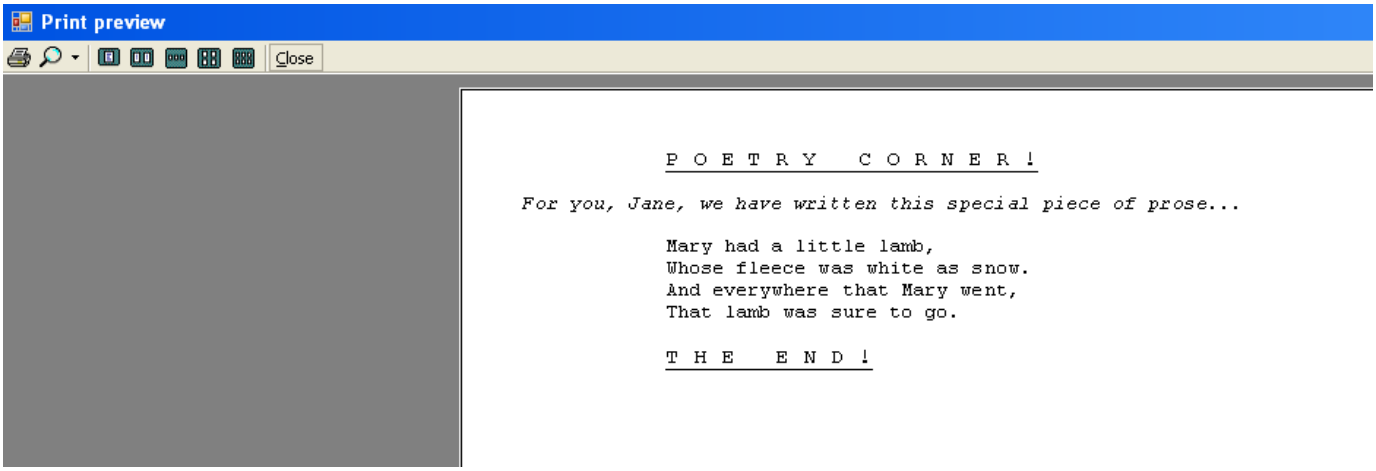
```
'Start the print preview dialog box off in maximised state.
PrintPreviewDialog1.WindowState = FormWindowState.Maximized
PrintPreviewDialog1.Document = PrintDocument1
'assign a document to Print Preview control
PrintPreviewDialog1.ShowDialog()
'display the PrintPreview dialog box
```

10. The PrintDocument1_PrintPage event (gained by double-clicking on the PrintDocument object added to the form) should contain the following code (and comments):

```
Dim MyTitleFont As New Font("Courier New", 14, FontStyle.Underline)
Dim MyHeaderFont As New Font("Courier New", 12, FontStyle.Italic)
Dim MyFont As New Font("Courier New", 12, FontStyle.Regular)
Dim MyFormat As String = "{0, -25}{1, -15}{2, -15}{3, -15}" '4 columns
Dim intTitleX, intX, intY, intFontHeight As Integer
intTitleX = 175
intX = 100   'set coordinates for first thing to print
intY = 50
intFontHeight = MyFont.GetHeight(e.Graphics)
'height of 1 line of font used - normal font, not title font.

e.Graphics.DrawString("P O E T R Y   C O R N E R !", _
        MyTitleFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.

intX = 50 'This is how far you want to indent from the edge of the page
```

```
intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
            'down the page as the report gets longer.
e.Graphics.DrawString("", MyFont, Brushes.Black, intX, intY) 'Blank line
intY = intY + intFontHeight

e.Graphics.DrawString("For you, " & strName & _
            ", we have written this special piece of prose...", _
MyHeaderFont, Brushes.Black, intX, intY) 'Put the title on the page.

intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
            'down the page as the report gets longer.
e.Graphics.DrawString("", MyFont, Brushes.Black, intX, intY) 'Blank line
intY = intY + intFontHeight

e.Graphics.DrawString("Mary had a little lamb,", _
MyFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.
intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
            'down the page as the report gets longer.

e.Graphics.DrawString("Whose fleece was white as snow.", MyFont, Brushes.Black, _
            intTitleX, intY) 'Next line
intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
            'down the page as the report gets longer.

e.Graphics.DrawString("And everywhere that Mary went,", MyFont, Brushes.Black, _
            intTitleX, intY) 'Next line
intY = intY + intFontHeight

e.Graphics.DrawString("That lamb was sure to go.", MyFont, Brushes.Black, _
            intTitleX, intY) 'Next line
intY = intY + intFontHeight

intY = intY + intFontHeight

e.Graphics.DrawString("T H E   E N D !", _
        MyTitleFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.
```

Output from this program will read:



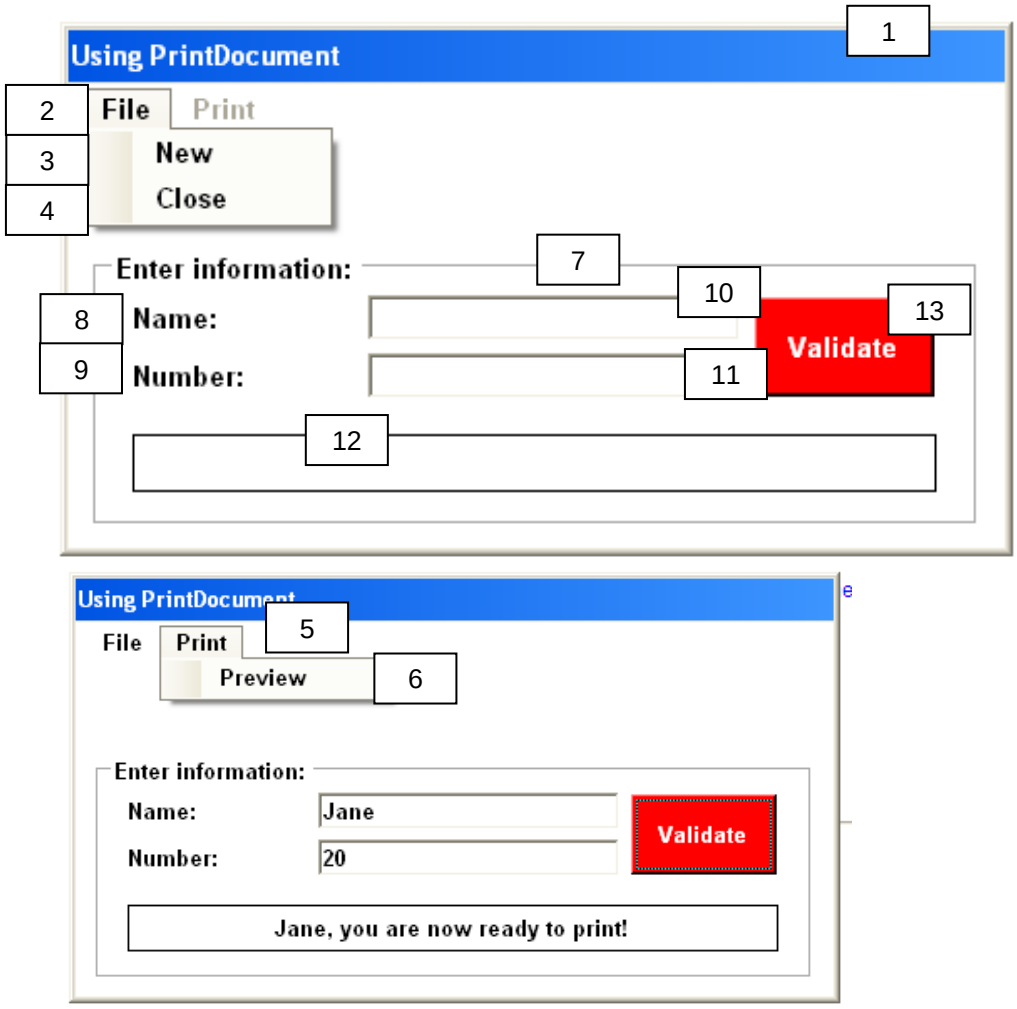
Coding for this program

```
Public Class Form1

    Dim strName As String

    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

        'Set the defaults for the project
        MenuStrip1.BackColor = Color.White

    End Sub

    Private Sub CloseToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
            Handles CloseToolStripMenuItem.Click

        Me.Dispose()
        End
    End Sub

    Private Sub PreviewToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
            Handles PreviewToolStripMenuItem.Click

        'Start the print preview dialog box off in maximised state.
        PrintPreviewDialog1.WindowState = FormWindowState.Maximized
        PrintPreviewDialog1.Document = PrintDocument1
        'assign a document to Print Preview control
        PrintPreviewDialog1.ShowDialog()
        'display the PrintPreview dialog box
    End Sub

    Private Sub PrintDocument1_PrintPage(sender As System.Object, e As _
            System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        Dim MyTitleFont As New Font("Courier New", 14, FontStyle.Underline)
        Dim MyHeaderFont As New Font("Courier New", 12, FontStyle.Italic)
        Dim MyFont As New Font("Courier New", 12, FontStyle.Regular)
        ' Dim MyFormat As String = "{0, -25}{1, -15}{2, -15}{3, -15}" '4 columns
        Dim intTitleX, intX, intY, intFontHeight As Integer
        intTitleX = 175
        intX = 100   'set coordinates for first thing to print
        intY = 50
        intFontHeight = MyFont.GetHeight(e.Graphics)
        'height of 1 line of font used - normal font, not title font.

        e.Graphics.DrawString("P O E T R Y   C O R N E R !", _
                MyTitleFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.

        intX = 50 'This is how far you want to indent from the edge of the page
        intY = intY + intFontHeight
       'reset Y for next line - Y increments and gets further down the page
       'as the report gets longer.
        e.Graphics.DrawString("", MyFont, Brushes.Black, intX, intY) 'Blank line
        intY = intY + intFontHeight
```

```
        e.Graphics.DrawString("For you, " & strName & _
                    ", we have written this special piece of prose...", _
                    MyHeaderFont, Brushes.Black, intX, intY) 'Put the title on the page.

        intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
              'down the page as the report gets longer.
        e.Graphics.DrawString("", MyFont, Brushes.Black, intX, intY) 'Blank line
        intY = intY + intFontHeight

        e.Graphics.DrawString("Mary had a little lamb,", _
        MyFont, Brushes.Black, intTitleX, intY) 'Put the first line on the page.
        intY = intY + intFontHeight

        e.Graphics.DrawString("Whose fleece was white as snow.", _
                MyFont, Brushes.Black, intTitleX, intY) 'Next line
        intY = intY + intFontHeight

        e.Graphics.DrawString("And everywhere that Mary went,", _
                MyFont, Brushes.Black, intTitleX, intY) 'Next line
        intY = intY + intFontHeight

        e.Graphics.DrawString("That lamb was sure to go.", _
                MyFont, Brushes.Black, intTitleX, intY) 'Next line
        intY = intY + intFontHeight

        intY = intY + intFontHeight

        e.Graphics.DrawString("T H E   E N D !", _
                MyTitleFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.

        'This code puts an image on the piece of paper.
        Dim bCorner As New Point(150, 645)
              'Establish where you want the image to appear on the page.

        Dim strImageName As String = Application.StartupPath 'Find where the program is stored
        strImageName = strImageName & "\maryHadALittleLamb.jpg" 'Concatenate the name of the iamge,
        'which you have previously stored in the Debug folder of the program.
        e.Graphics.DrawImageUnscaled(Image.FromFile(strImageName), bCorner) 'Draw it.

    End Sub

    Private Sub btnValidate_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnValidate.Click

        'Check to see that the user has entered something, and that the entry isn't too long.
        If txtName.Text.Length = 0 Then
            MsgBox("You must enter a name before continuing!", MsgBoxStyle.Critical, "ERROR!")
            txtName.Focus()
            Exit Sub
        End If

        If txtName.Text.Length > 20 Then
            MsgBox("You are allowed up to 20 characters for name!", MsgBoxStyle.Critical, "ERROR!")
            txtName.Text = ""
            txtName.Focus()
            Exit Sub
        End If

        strName = txtName.Text
```

```
        lblStatus.Text = strName & ", you are now ready to print!"
        PrintToolStripMenuItem.Enabled = True
    End Sub

    Private Sub NewToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
            Handles NewToolStripMenuItem.Click

        lblStatus.Text = ""
        txtName.Text = ""
        txtName.Focus()
        PrintToolStripMenuItem.Enabled = False
    End Sub
End Class
```

Exercise 14.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	MenuStrip1 (Print)
6	PreviewToolStripMenuItem
7	GroupBox1
8	Label1

Item	Object
9	Label2
10	TextBox1 (txtName)
11	TextBox2 (txtNumber)
12	Label1 (lblStatus)
13	Button1 (btnValidate)
14	(Hidden) PrintPreviewDialog
15	(Hidden) PrintDocument1

A program is needed to meet the following requirements:

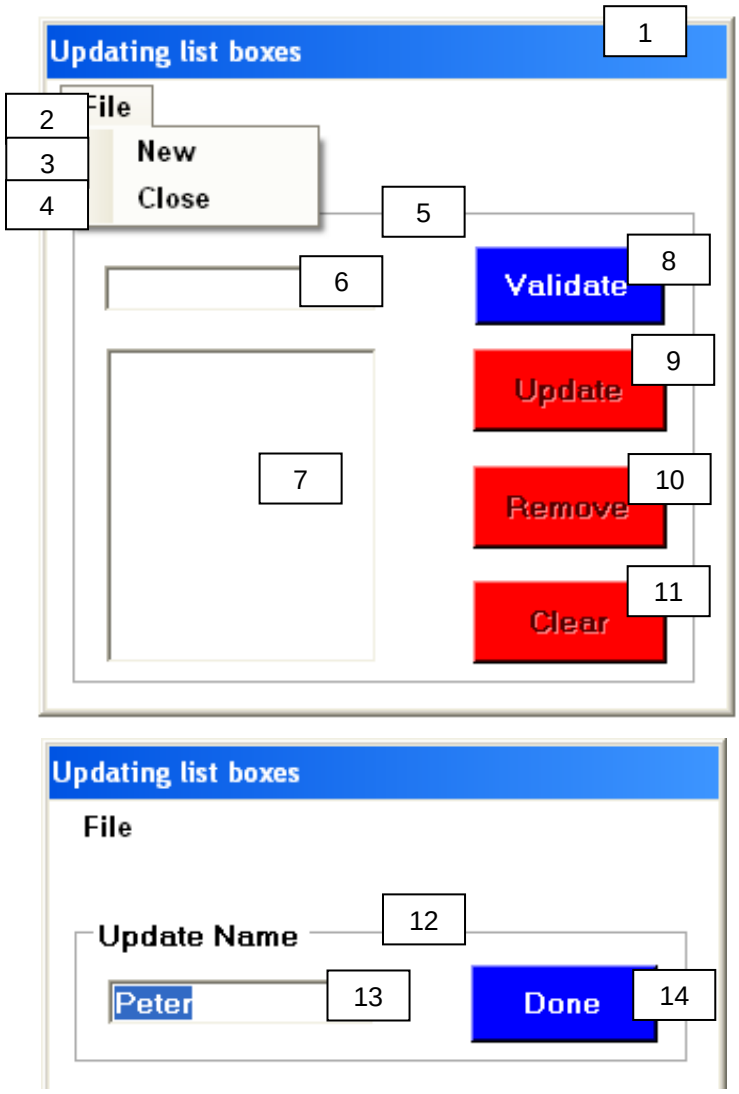
1. The menu strip should have two items at the top level: File and Print.
2. File should contain two items: New and Close.
3. Close should close the program.
4. New should reset the form to its original condition and all variables should be set to their original values. The Print option on the menu strip should be disabled.
5. When the program loads, everything should be visible on the form and the background colour of the menu strip set to whatever is used within the program for the background colour of the form. The Print option on the menu strip should be disabled.
6. The program needs these variable to be declared at the top of the program: strName as string, and intNumber as integer.
7. At design time, PrintDocument1 should be assigned as the "Document" in the PrintPreviewDialogue control.
8. The processing for btnValidate should be as follows:
    a. The user's entry into txtName should be validated to be within the length of 1 to 20 characters. Anything outside that range should be rejected with an appropriate error message and user should be exited from that procedure.
    b. Valid input should be stored in strName.
    c. The user's entry into txtNumber should be validated to be a number within the range of 1 to 20. Anything outside that range should be rejected with an appropriate error message and the user exited from that procedure.
    d. The message "*name*, you are now ready to print!" should be displayed in lblStatus, where *name* is the value stored in strName.
9. The Preview menu strip click event should contain the following code:

```
'Start the print preview dialog box off in maximised state.
PrintPreviewDialog1.WindowState = FormWindowState.Maximized
PrintPreviewDialog1.Document = PrintDocument1
'assign a document to Print Preview control
PrintPreviewDialog1.ShowDialog()
'display the PrintPreview dialog box
```

10. The PrintDocument1_PrintPage event (gained by double-clicking on the PrintDocument object added to the form) should contain the following code (and comments) in addition to any code that allows the document to print (see previous program):

```
Dim intCount, intAnswer As Integer

For intCount = 1 To 20

    intAnswer = intNumber * intCount
    e.Graphics.DrawString(intNumber & " * " & intCount & " = " & intAnswer & ".", _
    MyFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.
    intY = intY + intFontHeight 'reset Y for next line - Y increments and gets further
             'down the page as the report gets longer.

Next intCount
```

15.0 Specification for program developed in class



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	GroupBox1 (gbAdd)
6	TextBox1 (txtName)
7	ListBox1 (lstName)

Item	Object
8	Button1 (btnProcess)
9	Button2 (btnUpdate)
10	Button3 (btnRemove)
11	Button4 (btnClear)
12	GroupBox2 (gbUpdate)
13	TextBox2 (txtUpdatedName)
14	Button5 (btnDone)

A program is needed to meet the following requirements:

This program introduces how to insert data into list boxes.

1. The program needs the following variable to be declared at the top of the program:

```
Dim intPointer As Integer
```

2. The menu strip should have one item at the top level: File.
3. File should contain two items: New and Close.
4. Close should close the program.
5. New should reset the form to its original condition and all variables should be set to their original values. The subroutine CheckForEntries should be performed. (See item 7 in this specification).
6. When the program loads, everything except gbUpdate should be visible on the form and the background colour of the menu strip set to whatever is used within the program for the background colour of the form.
7. Also on the form load event, the subroutine CheckForEntries should be performed. The subroutine should be separate from other routines but should be positioned before the "End Class" statement of the program.

```
'This routine checks to make sure that there are some entries in the list box.
Dim intCount As Integer
intCount = lstName.Items.Count
If intCount > 0 Then
    'There are entries so enable all the buttons that apply to
    'the processing of the data in the list box.
    btnRemove.Enabled = True
    btnUpdate.Enabled = True
    btnClear.Enabled = True
Else
    'There are no entries in the list box so disable everything.
    btnRemove.Enabled = False
    btnUpdate.Enabled = False
    btnClear.Enabled = False
End If
```

8. The click event for btnProcess should (and in this order):
    a. Validate the entry made by the user in txtName as being in existence but being less than 21 characters. Any deviation from this should be reported to the user by use of a message box and the subroutine exited, with the text in txtName being removed and the cursor placed back there to await further input.
    b. Correct input should be added to lstName.
    c. The entry made in txtName should be removed and the cursor placed back there to await further input.
    d. Subroutine CheckForEntries should be performed.

9. The click event for btnUpdate should (and in this order):
    a. Set the accept button for the form to be btnDone.

b. Check to ensure that the user has selected a name in lstName by examining the SelectedIndex property of lstName to ensure that it isn't equal to -1. If it is, then a message box should be shown with a suitable error message and the subroutine exited.
c. If the entry is valid, then gbAdd should be set to invisible.
d. gbUpdate should be set to be visible and enabled.
e. The position of the selected name should be stored in intPointer:

```
'Set the pointer to equal the current place in the list box
intPointer = lstName.SelectedIndex
```

f. The selected name should be placed into the text box ready for the user to edit it.

```
'Put the text to be updated in the text box of the update groupbox.
txtUpdatedName.Text = lstName.Items.Item(intPointer)
```

g. The cursor should be placed in txtUpdatedName.
h. The background colour of btnUpdate should be set to red.

10. The click event of btnDone should (and in this order):
    a. Validate the user's entry in txtUpdatedName in the same way as for txtName. Any erroneous input should be identified with a message box, and then the text copied back into the text box and the cursor placed into the text box.

```
txtUpdatedName.Text = lstName.SelectedItem
txtUpdatedName.Focus()
```

    b. Assuming correct input, the program should now remove the old entry from lstName and replace it with the new entry from txtUpdatedName, but in the same place as the old name, using intPointer to indicate where in the list box the entry should be positioned.

```
'Remove the old item
lstName.Items.RemoveAt(lstName.SelectedIndex)
'Put the new item in the old one's place
lstName.Items.Insert(intPointer, txtUpdatedName.Text)
```

    c. The accept button of the form should be set to btnProcess.
    d. Set gbUpdate to be invisible.
    e. Set gbAdd to be visible.
    f. Put the cursor in txtName to await user entry.

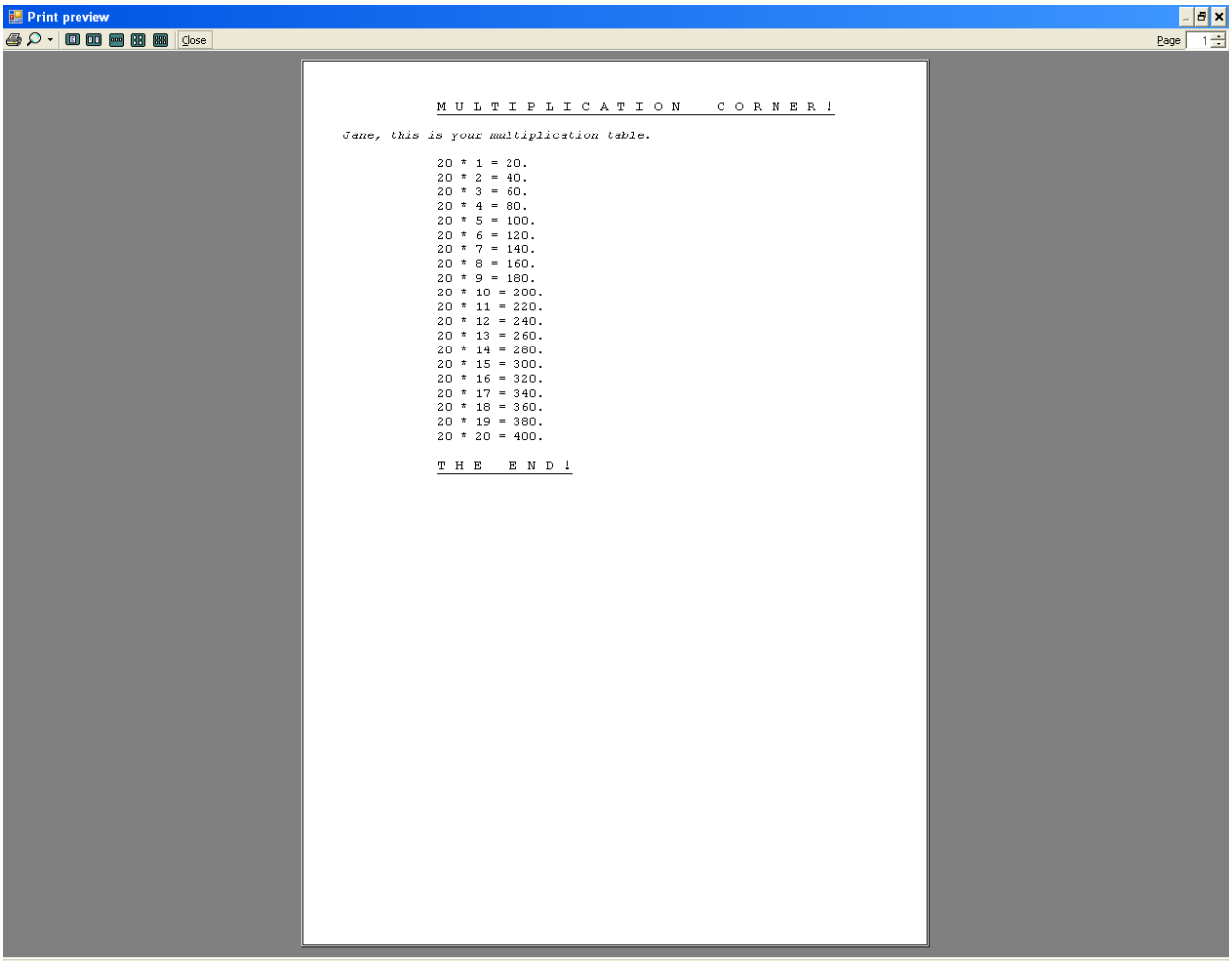
11. On the Click event of btnRemove (in this order):
    a. Check to ensure that the user has selected a name in lstName by examining the SelectedIndex property of lstName to ensure that it isn't equal to -1. If it is, then a message box should be shown with a suitable error message and the subroutine exited.
    b. Assuming that an entry has been selected, remove it.

```
lstName.Items.RemoveAt(lstName.SelectedIndex)
```

12. On the click event of btnClear (in this order):
    a. Ask the user if they really want to delete everything via a YesNo message box.
    b. If the answer is yes, then clear the list box of all entries.
    c. Perform the subroutine CheckForEntries.
    d. Put the cursor into txtName.

13. On the click event of lstName:
    a. Change the background colour of btnUpdate from red to blue.
    b. Change the background colour of btnProcess from blue to red.

Output from this program will read:



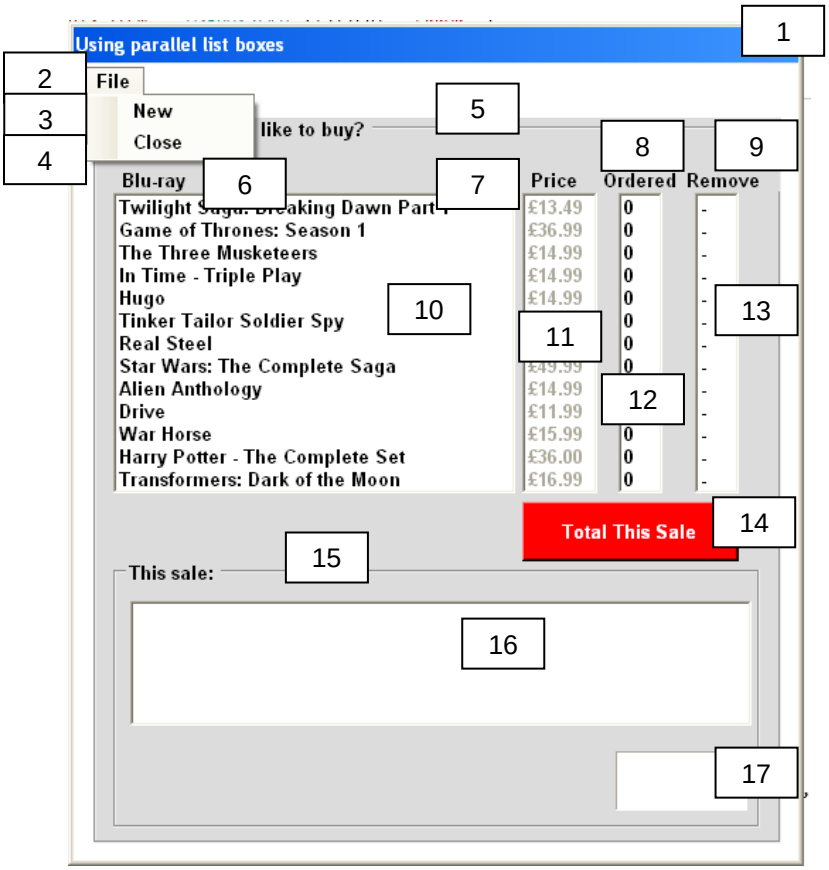
Coding for this program

```
Public Class Form1

    Dim intPointer As Integer

    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

        'Set the defaults for the program.
        MenuStrip1.BackColor = Color.White
        CheckForEntries()
    End Sub

    Private Sub CloseToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
              Handles CloseToolStripMenuItem.Click

        'End the program
        Me.Dispose()
        End
    End Sub

    Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnProcess.Click

        'Check that the user has entered something with a length in the
        'range of 1 to 20 characters.

        If (txtName.Text.Length = 0) Or (txtName.Text.Length > 20) Then
            MsgBox("Your entry should be text up to 20 characters in length!", _
                   MsgBoxStyle.Critical, "ERROR!")
            txtName.Text = ""
            txtName.Focus()
            Exit Sub
        End If

        'If we get to here, we're valid.
        'Add the name to the list box
        lstName.Items.Add(txtName.Text)

        'Put the focus back in the cleared text box
        txtName.Text = ""
        txtName.Focus()

        CheckForEntries()
    End Sub

    Private Sub CheckForEntries()

        'This routine checks to make sure that there are some entries in the list box.
        Dim intCount As Integer
        intCount = lstName.Items.Count
        If intCount > 0 Then
            'There are entries so enable all the buttons that apply to
            'the processing of the data in the list box.
            btnRemove.Enabled = True
            btnUpdate.Enabled = True
            btnClear.Enabled = True
```

```
        Else
            'There are no entries in the list box so disable everything.
            btnRemove.Enabled = False
            btnUpdate.Enabled = False
            btnClear.Enabled = False
        End If
    End Sub

    Private Sub btnUpdate_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnUpdate.Click

        'Change the AcceptButton of the form to btnUpdate
        Me.AcceptButton = btnDone

        'The user must have selected something in the list box
        'to be allowed to do this, so check to make sure
        'that they have.

        If lstName.SelectedIndex > -1 Then
        Else
            MsgBox("You must select a name from the list before continuing!", _
                   MsgBoxStyle.Critical, "ERROR!")
            Exit Sub
        End If

        'Disappear the current group box so that the user can't click something in error.
        gbAdd.Visible = False

        'Set the pointer to equal the current place in the list box
        intPointer = lstName.SelectedIndex

        'Put the text to be updated in the text box of the update groupbox.
        txtUpdatedName.Text = lstName.Items.Item(intPointer)

        'Enable the update list box
        gbUpdate.Visible = True
        gbUpdate.Enabled = True
        txtUpdatedName.Focus()

        btnProcess.BackColor = Color.Blue
        btnUpdate.BackColor = Color.Red
    End Sub

    Private Sub btnDone_Click(sender As System.Object, e As System.EventArgs) Handles btnDone.Click

        'Update the entry in the list box with the new information,
        'after it has been validated by the same rules.
        If (txtUpdatedName.Text.Length = 0) Or (txtUpdatedName.Text.Length > 20) Then
            MsgBox("Your entry should be text up to 20 characters in length!", _
                   MsgBoxStyle.Critical, "ERROR!")
            txtUpdatedName.Text = lstName.SelectedItem
            txtUpdatedName.Focus()
            Exit Sub
        End If

        'Remove the old item
        lstName.Items.RemoveAt(lstName.SelectedIndex)
        'Put the new item in the old one's place
        lstName.Items.Insert(intPointer, txtUpdatedName.Text)
```

```
        'Change the AcceptButton back to btnProcess
        Me.AcceptButton = btnProcess
        gbUpdate.Visible = False
        gbAdd.Visible = True
        txtName.Focus()
    End Sub

    Private Sub btnRemove_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnRemove.Click

        'Check to make sure that the user has selected something to delete
        If lstName.SelectedIndex > -1 Then
        Else
            MsgBox("You must select a name from the list before continuing!", _
                   MsgBoxStyle.Critical, "ERROR!")
            Exit Sub
        End If

        lstName.Items.RemoveAt(lstName.SelectedIndex)
    End Sub

    Private Sub btnClear_Click(sender As System.Object, e As System.EventArgs) _
              Handles btnClear.Click

        'Check to make sure that this is what the user wants.
        Dim intResponse As Integer
        intResponse = MsgBox("Are you sure you want to delete everything?", _
                    MsgBoxStyle.YesNo, "Are you sure?")
        If intResponse = vbYes Then 'Clear everything down
            lstName.Items.Clear()
            MsgBox("Everything deleted!", MsgBoxStyle.OkOnly, "Gone!")
        End If

        'Check to make sure that the buttons are in the right state - either
        'enabled because there are still entries in the list box
        'or disabled because the list box is empty.
        CheckForEntries()

        'Put the cursor back into the right text box.
        txtName.Focus()
    End Sub

    Private Sub NewToolStripMenuItem_Click(sender As System.Object, e As System.EventArgs) _
              Handles NewToolStripMenuItem.Click

        'Get rid of everything
        lstName.Items.Clear()
        txtName.Text = ""
        txtName.Focus()
        CheckForEntries()
    End Sub

    Private Sub lstName_Click(sender As Object, e As System.EventArgs) Handles lstName.Click

        'Change the background colour of the button to update.
        btnUpdate.BackColor = Color.Blue
        btnProcess.BackColor = Color.Red
    End Sub
End Class
```

Exercise 15.1



Item	Object
1	Form1
2	MenuStrip1 (File)
3	NewToolStripMenuItem
4	CloseToolStripMenuItem
5	GroupBox1
6	Label1 (“Blu-ray”
7	Label2 (“Price”)
8	Label3 (“Ordered”)
9	Label4 (“Remove”)

Item	Object
10	ListBox1 (lstTitle)
11	ListBox2 (lstPrice)
12	ListBox3 (lstOrdered)
13	ListBox4 (lstRemove)
14	Button1 (btnTotal)
15	GroupBox2
16	ListBox5 (lstSale)
17	Label5 (lblAmountOwing)

A program is needed to meet the following requirements:
This program practices the art of synchronising list box pointers.

1. The program needs the following variables to be declared at the top of the program:

```
Dim decTotal As Decimal 'Used to add up the total of how much the customer owes.
Dim decPrice As Decimal
Dim intQuantity As Integer

Dim MyFormat As String = "{0, -40} {1, 5} {2, 5} {3, 6}"
'Title, Price, Quantity, Total
```

2. The menu strip should have one item at the top level: File.
3. File should contain two items: New and Close.
4. Close should close the program.
5. New should reset the form to its original condition and all variables should be set to their original values. lstOrdered should have all its values set back to 0:

```
Dim intPointer As Integer
For intPointer = 0 To lstTitle.Items.Count - 1
    lstOrdered.Items.Add("0")
Next
```

   Also, ensure that no item is selected in the Titles list box by setting its SelectedIndex property to -1.
6. When the program loads, everything should be visible on the form and the background colour of the menu strip set to whatever is used within the program for the background colour of the form.
7. The load event of the form should set the background colour of the menu strip to be the same colour as for the form.
8. The blu-ray titles and prices within lstTitle and lstPrice should be hard-coded within their respective Items properties.
9. lstRemove should have an equivalent number of "-" hardcoded within its Items property.
10. On the Click event of lstTitle:
    a. The following code is necessary to ensure that the index of the selected title is retained for use with the other list boxes.

```
'Work out which blu-ray has been selected.
Dim intPointer As Integer
intPointer = lstTitle.SelectedIndex
```

    intPointer now contains the value of "where we are at" for each of the list boxes in the program.
    b. After that, the program must query how many of that particular title have already been ordered, by looking at the corresponding value in lstOrdered.

```
'Get how many we had ordered of this blu-ray in the first instance.
Dim intQuantity As Integer
intQuantity = lstOrdered.Items.Item(intPointer)
```

intQuantity now contains the value of how many of that title have been ordered already.

c. The program must now remove the old value of how many have been ordered from the "Ordered" list.

```
'Remove the old amount of blu-rays
lstOrdered.Items.RemoveAt(lstTitle.SelectedIndex)
```

d. The user wants to add one to the order for that particular title so we must account for that:

```
'Add one to how many we've ordered
intQuantity += 1
```

e. We now update the quantity in the Ordered list box by inserting the new quantity ordered in at the appropriate place (place stored in intPointer, quantity in intQuantity):

```
'Insert the updated number of blu-rays.
lstOrdered.Items.Insert(intPointer, (intQuantity))
```

11. On the click event of lstRemove:

a. For this process we must ascertain upon which title the user wishes to decrease the order:

```
'Work out which blu-ray has been selected.
Dim intPointer As Integer
intPointer = lstRemove.SelectedIndex
```

intPointer now contains the value of "where we are at" for each of the list boxes in the program.

b. After that, the program must query how many of that particular title have already been ordered, by looking at the corresponding value in lstOrdered.

```
'Get how many we had ordered of this blu-ray in the first instance.
Dim intQuantity As Integer
intQuantity = lstOrdered.Items.Item(intPointer)
```

intQuantity now contains the value of how many of that title have been ordered already.

c. After that the program must check to see that the selected title has an order placed already, and if not display an appropriate error message. If an order is in existence, then one must be deducted from the quantity ordered and the list box for Orders updated:

```
'check to make sure that there is something there to remove
If intQuantity > 0 Then
    'Remove the old amount of blu-rays
    lstOrdered.Items.RemoveAt(lstRemove.SelectedIndex)

    intQuantity -= 1
```

```
        'Insert the updated number of blu-rays.
        lstOrdered.Items.Insert(intPointer, (intQuantity))
    Else
        MsgBox("There are no orders for this item!", MsgBoxStyle.Critical, "ERROR!")
    End If
```

12. On the click event of btnTotal:
    a. Clear any current contents of lstSale.
    b. Set the variable decTotal to equal 0.
    c. For each title, the number of orders should be examined and if that number is greater than 0 then the number of copies of that particular title should be multiplied by the corresponding price in lstPrice and the total for that sum stored in a very local variable, decNetPrice. decNetPrice should be added to decTotal. The appropriate item should be displayed in the list box lstSale, using MyFormat – title, followed by price, followed by quantity ordered, followed by net price.

```
'Examine each of the objects and if there are any orders for this object,
‘process it.
For intPointer = 0 To lstTitle.Items.Count - 1

    'Check to see if the user has ordered any of these blu rays by examining the
    'value stored in the Ordered column.

    If lstOrdered.Items.Item(intPointer) > 0 Then

        'Save the price of this blu-ray for use in mathematical calculation.
        decPrice = lstPrice.Items.Item(intPointer)

        'Get how many orders we'd already got for this item
        intQuantity = lstOrdered.Items.Item(intPointer)

        'Calculate how much the sale is worth for the list box
        Dim decNetPrice As Decimal = decPrice * intQuantity

        'Display the total transaction for the list box.
        lstSale.Items.Add(String.Format(MyFormat, _
             lstTitle.Items.Item(intPointer), _
             Format(decPrice, "Currency"), _
              intQuantity, Format(decNetPrice, "Currency")))

        'Accumulate the overall total for the label when the user
        ‘clicks total button.
        decTotal += decNetPrice
    End If
Next intPointer
```

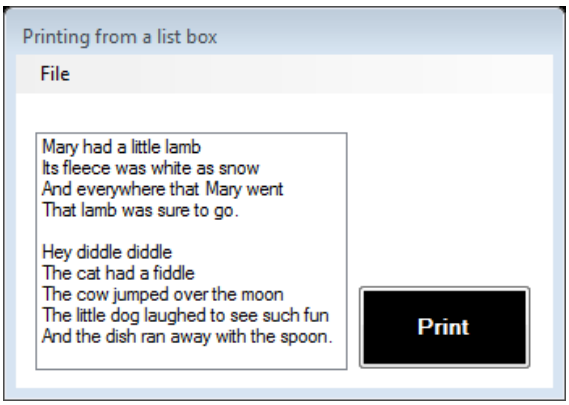
    d. Display decTotal in currency format in lblAmountOwing.
    e. Ensure that no item in lstSale is selected.

Example programs – 16. Printing from a ListBox

Lecturer: Jane Fletcher

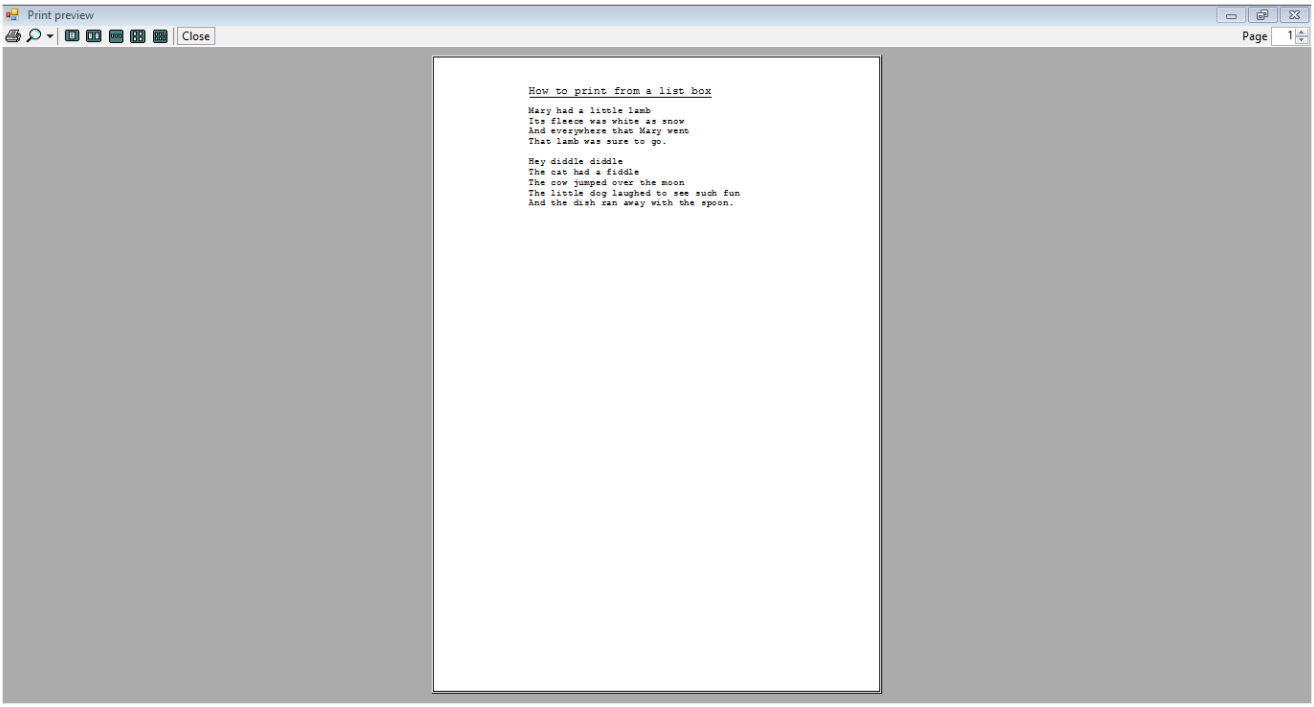
In order to print out the items from a ListBox control, use the For Each … Next construct. In the sample code below there are two nursery rhymes typed directly into the Items collection of the ListBox object lstRhymes. When the user clicks the Button object btnPrint, the PrintPreviewDialog object is referenced, which calls the PrintDocument1.PrintPage function.

The output from the program looks like this:



This is the form once the program is running. The File MenuStrip item contains only "End", which stops the program running. When the "Print" button is pressed, the PrintPreviewDialog object produces the following:

The code to make this happen is shown on the next page.



```
Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click

    'Close the project
    Me.Dispose()
    End
End Sub

Private Sub btnPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
                Handles btnPrint.Click

    'Preview the PrintPage object's printpage.
    PrintPreviewDialog1.WindowState = FormWindowState.Maximized
    PrintPreviewDialog1.Document = PrintDocument1
    'assign a document to Print Preview control
    PrintPreviewDialog1.ShowDialog()
    'display the PrintPreview dialog box

End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As  _
            System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
    Dim MyTitleFont As New Font("Courier New", 14, FontStyle.Underline)
    Dim MyFont As New Font("Courier New", 12, FontStyle.Regular)
    Dim intTitleX, intX, intY, intFontHeight As Integer
    intTitleX = 175
    intX = 100   'set coordinates for first thing to print
    intY = 50
    intFontHeight = MyFont.GetHeight(e.Graphics)
    'height of 1 line of font used - normal font, not title font.

    e.Graphics.DrawString("How to print from a list box", _
            MyTitleFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.

    intY = intY + intFontHeight 'reset Y for next line - Y increments and gets
    'further down the page as the report gets longer.
    e.Graphics.DrawString("", MyFont, Brushes.Black, intTitleX, intY) 'Blank line
    intY = intY + intFontHeight

    Dim strItem As String
          'Set up this string variable to store the contents of the list box.

    For Each strItem In lstRhymes.Items 'This piece of code takes each
        'item that is stored in the Items list within the listbox object
        e.Graphics.DrawString(strItem, _
                MyFont, Brushes.Black, intTitleX, intY) 'Put the title on the page.
        intY = intY + intFontHeight
    Next

End Sub
```

The code in bold is the code required to take each item from the ListBox object's Items collection and print it.

Now create a program to print from the list box in Exercise 15.1.

Level 3: BTEC National
Credit value: 10
Guided learning hours: 60

- **Aim and purpose**

  This unit aims to enable learners to develop the skills and understanding required to design and develop event driven applications.

- **Unit introduction**

  Event driven programming is a very flexible way of allowing programs to respond to many inputs or events. Unlike traditional programming, where the control flow is determined by the program structure, the control flow of event driven programs is largely driven by external events. Typically, event loops are pre-programmed to continually look for information to process.

  This unit allows learners to become familiar with the underpinning concepts of event driven programming and subsequently to develop particular skills in an event driven language. The unit starts by looking at the features of event driven programming, explores the tools and techniques used in their development and takes learners through design and program development. Learners will use a structured approach to the design and development of applications, ensuring the solution is well documented and tested thoroughly against the original user requirement.

  Event handling features in many languages including Visual Basic, Visual Basic for Applications and many other systems.

- **Learning outcomes**

  On completion of this unit a learner should:

  1. Understand the features of event driven programming
  2. Be able to use the tools and techniques of an event driven language
  3. Be able to design event driven applications
  4. Be able to implement event driven applications.

Unit content

1 Understand the features of event driven programming

Key features: service oriented; time driven; event handlers; trigger functions; events eg mouse, keyboard, HTML object, form, user interface; event loops; flexibility; suitability for graphical interfaces; simplicity of programming; ease of development
Examples: operating systems as event driven systems; Graphical User Interfaces (GUIs)
Programming languages: eg Visual Basic (VB), Visual Basic for Applications (VBA), Coldfusion; Integrated Development Environments (IDEs)

2 Be able to use the tools and techniques of an event driven language

Triggers: eg key press, alarm, system event, touch screen event, mouse click
Tools and techniques: eg use of tool boxes and controls, selection, loops, event handlers, triggers, objects and object properties, menus; debugging tools
Variables: declaring variables; scope of variables; constants; data types

3 Be able to design event driven applications

Specification: input; output; processes; user need; purpose
Design: selecting and assigning properties to screen components; data storage; event procedures and
descriptions; appropriate ways of representing the processing tasks

4 Be able to implement event driven applications

Creation of application: use of development environment; debugging; data validation; error handling and reporting
Programming language syntax: eg selecting, declaring and initialising variable and data structure types and sizes
Constructs: selection eg if … then … else, CASE; iteration eg while … do, repeat … until
Programming standards: eg use of comments; code layout; indentation
Testing: test strategy; test plan structure eg test, date, expected result, actual result, corrective action; error messages; specialist software tools eg debug
Review: against specifications requirements; interim reviews
Documentation: user; technical

Assessment and grading criteria

In order to pass this unit, the evidence that the learner presents for assessment needs to demonstrate that they can meet all the learning outcomes for the unit. The assessment criteria for a pass grade describe the level of achievement required to pass this unit.

To achieve a pass grade the evidence must show that the learner is able to:		**To achieve a merit grade the evidence must show that, in addition to the pass criteria, the learner is able to:**		**To achieve a distinction grade the evidence must show that, in addition to the pass and merit criteria, the learner is able to:**	
P1	explain the key features of event driven programs	**M1**	discuss how an operating system can be viewed as an event driven application	**D1**	evaluate the suitability of event driven programs for non-graphical applications
P2	demonstrate the use of event driven tools and techniques	**M2**	give reasons for the tools and techniques used in the production of an event driven application		
P3	design an event driven application to meet defined requirements				
P4	implement a working event driven application to meet defined requirements				
P5	test an event driven application	**M3**	analyse actual test results against expected results to identify discrepancies	**D2**	evaluate an event driven application
P6	create onscreen help to assist the users of a computer program.	**M4**	create technical documentation for the support and maintenance of a computer program.		

Assessment

It is suggested that this unit is assessed using four assignments as summarised in the *Programme of suggested assignments* table.

Finding a scenario which covers all aspects of all criteria is difficult, but the one suggested is acceptable. It places the user in a role which is at an acceptable level for their experience, which is important when devising assignments. Some of the evidence required to complete the assignments could be naturally occurring within their work for other units within the qualification, or for other courses they are undertaking, and tutors are encouraged to use such evidence. Evidence produced for this assignment can be used towards the evidence required for other criteria in this unit.

In order to gain a pass grade, learners must meet all of the pass criteria.
For P1, learners must explain the features required to implement a given design. This refers to the features section of the unit content for learning outcome 1. In order to achieve this criterion, learners must describe the features clearly and logically, showing they have recognised the underpinning principles and in particular, the reasons why triggers and timing are important. A presentation or leaflet would be a suitable form of evidence.
For P2, learners must show that they are able to use event driven programming tools and techniques, including those listed in the unit content. A presentation would be a suitable form of evidence.
For P3, learners must design an event driven program. The program only needs to be basic, as suited to the level of learners, but obviously this is at the discretion of the tutor and the individual learners. The design should be clear and have no obvious errors.
For P4, learners must create the program they worked on for P3. This program should be fully functional, and fulfil the design aims set down.
For P5, learners must develop and apply an appropriate test plan for the program they worked on for P4. The test plan should test functionality and demonstrate that the program fulfils the design aims and other requirements. Evidence is likely to be in the form of a short report on the test plan and results, illustrated with screen grabs.
For P6, learners must produce onscreen help for an event driven program. The help screens must be coherent and laid out according to the standards that learners have previously been taught. It is up to the tutor at this stage whether they wish to have learners use the work they have produced for P4 and P5, or to give learners a generic event driven program for which to write the appropriate help.

In order to gain a merit, learners must achieve all the pass criteria, and all of the merit criteria.
For M1, learners must discuss how an operating system can be viewed as an event driven application. This will probably be the operating system that learners are working with but tutors may choose a different one if they wish. As with P1, evidence should be a poster, leaflet or short report but a presentation can also be used if the learner or tutor would prefer.
For M2, learners must justify their choice of tools and techniques used in the production of the event driven application created in P3. Evidence should be a short report, or similarly detailed presentation.
For M3, learners must analyse the results of their testing in P6. The analysis should compare expected to actual results to identify discrepancies. It would also be expected that learners would suggest what actions should be taken to resolve any problems shown up by the testing. Evidence for this criterion should be a short report. This could be an extension of the P6 report.
For M4, learners will create technical documentation for the support and maintenance of a computer program. As with P6, it is up to the tutor whether they wish to have learners use the work

they have carried out for P4 and P5, or to give learners a generic event driven program. The documentation must be coherent and laid out according to the standards that learners have been taught previously.

In order to gain a distinction grade, learners must achieve all of the pass and merit criteria and both the distinction criteria.
For D1, learners will evaluate the suitability of event driven programs for non-graphical applications. Evidence for this criterion should be a short report, or similarly detailed presentation.
For D2, learners will evaluate an event driven application. There is the option here for the learner to review their P5 work, or to be given a generic program to review. Alternatively, the tutor could give learners the work of one of their peers to review. Evidence should be a short report, or similarly detailed presentation.

Indicative reading for learners

Textbooks

Balena F – *Programming Microsoft Visual Basic 6* (Microsoft Press US, 1999) ISBN-10: 0735605580, ISBN-13: 978-0735605589
Bond M, Law D, Longshaw A, Haywood D and Roxburgh P – *Sams Teach Yourself J2EE in 21 Days, 2nd Edition* (Sams, 2004) ISBN-10: 0672325586, ISBN-13: 978-0672325588
Palmer G – *Java Event Handling* (Prentice Hall, 2001) ISBN-10: 0130418021, ISBN-13: 978-0130418029
Longshaw J and Sharp J – *Visual J#.NET Core Reference* (Microsoft Press US, 2002) ISBN-10: 0735615500, ISBN-13: 978-0735615502
Suddeth J – *Programming with Visual Studio.NET 2005* (Lulu.com, 2006) ISBN-10: 1411664477, ISBN-13: 978-1411664470
Troelsen A – *Pro C# 2005 and the.NET 2.0 Platform, 3rd Edition* (Apress US, 2004) ISBN-10: 1590594193, ISBN-13: 978-1590594193

Websites

eventdrivenpgm.sourceforge.NET
www.vbexplorer.com/VBExplorer/VBExplorer.aspte
www.vbwm.com

Appendix 2 – Glossary of Terms

Lecturer: Jane Fletcher

This unit relies on your understanding of some fairly complex programming terminology. As the unit progresses and your learning increases, you should keep this glossary up to date to that you can refer to it when you need to.

Term	Definition
LO 1 : Understand the features of event driven programming	
Service-oriented	
Time-driven	
Event handlers	
Trigger functions	
Events	
Examples of events:	
Mouse	
Keyboard	
HTML object	
Form	
User interface	
Event loops	
Flexibility	
Suitability for GUI	
Simplicity for programming	
Ease of development	
LO 2 : Be able to use the tools and techniques of an event driven language	
Triggers	
Examples of triggers:	
Key press	
Alarm	
System event	
Touch screen event	
Mouse click	

Appendix 2 – Glossary of Terms

Lecturer: Jane Fletcher

Term	Definition
Tools and techniques:	
Tool boxes and controls	
Selection	
Loops	
Event handlers	
Triggers	
Objects	
Object properties	
Menus	
Debugging tools	
Variables:	
Declaration (name, type)	
Scope	
Constants	
Data types	
LO 3 : Be able to design event driven applications	
Specification:	
Input	
Output	
Processes	
User need	
Purpose	
Design:	
Selecting and assigning properties	
Data storage	
Event procedures	
LO 4 : Be able to implement event driven applications	
Creation of application:	
IDE	
Debugging (how?)	
Data validation (what and how)	
Error handling and reporting (what and how)	

Appendix 2 – Glossary of Terms

Lecturer: Jane Fletcher

Term	Definition
Construct (definition)	
Construct (examples)	
Documentation:	
User	
Technical	
GENERAL (for your use)	

Appendix 3 – Log of completed exercises

Lecturer : Jane Fletcher

This section is to enable you to keep a record of each of the exercises in this book. You will be directed by your lecturer when you should be doing them, but you must make sure that you keep up to date because if you don't you are going to struggle with the final assignment.

The key thing with programming is practice, **practice**, **PRACTICE**!

But even more importantly, **ENJOY IT**!

Appendix 3 – Log of completed exercises

Lecturer : Jane Fletcher

LOG OF PROGRAMMING EXERCISES				
Program	**Date Started**	**Date Finished**	**Checked?**	**Comments**
1.0. Done in class				
1.1. Exercise 1				
2.0. Done in class				
2.1. Exercise 1				
3.0. Done in class				
3.1. Exercise 1				
3.2. Exercise 2				
4.0. Done in class				
4.1. Exercise 1				
4.2. Exercise 2				
5.0. Done in class				
5.1. Exercise 1				
5.2. Exercise 2				
6.0. Done in class				
6.1. Exercise 1				
7.0. Done in class				
7.1. Exercise 1				
8.0. Done in class				
8.1. Exercise 1				
8.2. Exercise 2				
9.0. Done in class				
9.1. Exercise 1				
10.0. Done in class				
10.1. Exercise 1				
11.0. Done in class				
11.1. Exercise 1				
12.0. Done in class				
12.1. Exercise 1				
13.0. Done in class				
13.1. Exercise 1				
14.0. Done in class				
14.1. Exercise 1				
15.0. Done in class				
15.1. Exercise 1				
16.0. Done in class				
16.1. Exercise 1				

INDEX

Lecturer : Jane Fletcher

INDEX

Lecturer : Jane Fletcher

INDEX

Lecturer : Jane Fletcher

P

R

S

T

U

V